INTEGRATION OF BUILDING PRODUCT MODELS WITH FIRE SIMULATION SOFTWARE

A thesis submitted in partial fulfilment of the
requirements for the degree
of
Doctor of Philosophy in Fire Engineering
at the
University of Canterbury,
Christchurch, New Zealand.

by

Michael John Spearpoint

November 2005

ABSTRACT

There is considerable interest within the construction industry to develop models that are able describe the whole life-cycle of a building in an electronic form. Such models would allow for the sharing of building information across the wide range of industry disciplines and lead to efficiencies in the design and construction of buildings. This thesis examines the technologies available and specifically within the context of fire engineering.

Database methods are used to create a repository of fire growth information which can be accessed through web pages or client applications. The IFC Model has emerged as an internationally agreed building model and this thesis investigates its applicability to fire engineering. A suite of software applications have been developed that interpret IFC documents in a form that can be imported into fire simulation models. The thesis discusses the limitations of the current IFC model for use by fire engineers, the challenges in developing IFC interpretation software that can be successfully integrated with the range and complexity of fire simulation software and suggests where future work should be directed to overcome these concerns.

ACKNOWLEDGEMENTS

I would like to thank Dr. Charley Fleischmann and Dr. Bruce Deam for acting as my supervisors during this thesis and I would also like to give a special mention to Prof. Andy Buchanan for his ongoing encouragement. Colleen Wade of BRANZ kindly provided access to the BRANZFIRE fire simulation software. I am grateful to the Department of Civil Engineering, University of Canterbury for granting me the resources to complete this thesis. I would also like to acknowledge the New Zealand Fire Service Commission for their generosity whilst I carried out this research and for their continued sponsorship of the Fire Engineering programme at the University of Canterbury. Finally, thanks to Fiona McCartin for her support.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

FONT STYLES

The following font style conventions are used in this thesis to help the reader distinguish between different parts of the text:

- Thesis text (Times New Roman, 12pt)
- `Source code (Courier New, 11pt)`
- XML script e.g. <xml_tag attribute="attr">value</xml_tag> (Times New Roman, 11pt, coloured font)
- *Names of windows, buttons, menu items etc. (Microsoft Sans Serif, 12pt, italic)*
- **Computer program / application / operating system (Tahoma, 11pt, bold)**
- Filename / database (Tahoma, 11pt)
- *IFC entities (Times New Roman, 12pt, italic)*
- Hyperlink (Times New Roman, 12 pt, underline, font colour: blue)

ABBREVIATIONS

AEC	Architecture, Engineering and Construction
AHJ	Authority Having Jurisdiction
AS	Australian Standard
BCA	Building Code of Australia
BIA	Building Industry Authority
BIM	Building Information Modelling (or Model)
BLIS	Building Lifecycle Interoperable Software
BRANZ	Building Research Association of New Zealand
CAD	Computer Aided Design (Drafting)
CFD	Computational Fluid Dynamics
CIC	Computer-Integrated Construction
CSS	Cascading Style Sheets
CSV	Comma-Separated Values
DOS	Disk Operating System
DTD	Document Type Definition
DXF	Data Exchange Format
FAST	**Fire And Smoke Transport**
FDMS	**Fire Data Management System**
FDS	Fire Dynamics Simulator
FEMA	Federal Emergency Management Agency
FHC	Fire Hazard Categories (as defined by C/AS1, see BIA, 2001)
FM	Facilities Management
FRR	Fire Resistance Ratings (as defined by C/AS1, see BIA, 2001)
GIS	Geographical Information Systems
HTML	Hyper-Text Mark-up Language
HRR	Heat Release Rate, alternatively RHR
HVAC	Heating, Ventilation and Air Conditioning
IAI	International Alliance for Interoperability
IE	[Microsoft] **Internet Explorer**
IFC	Industry Foundation Classes
ISO	International Organization for Standardization

IT	Information Technology
LAN	Local Area Network
MS	Microsoft
NCS/BCS	National Conference of States on Building Codes and Standards
NIST	National Institute of Standards and Technology
NZS	New Zealand Standard
PC	Personal Computer
PDF	[Acrobat] Portable Document Format
PSD	Property Set Definition
RHR	Rate of Heat Release, alternatively HRR
RTI	Response Time Index
SFI	Spread of Flame Index
SGML	Standard Generalized Markup Language
SFPE	Society of Fire Protection Engineers
STEP	STandard for the Exchange of Product model data
URL	Uniform Resource Locator
VR	Virtual Reality
W3C	World Wide Web Consortium
WAN	Wide Area Network
XML	eXtensible Markup Language
XSD	XML Schema Definition
XSL	eXtensible Stylesheet Language
XSLT	XSL Transformations

TERMINOLOGY

This thesis uses a set of terminology that may be unfamiliar to the reader. Much of this terminology has developed within the information technology industry and specifically in the XML, IFC and object-oriented programming fields. The following list is provided to assist the reader and further details on many of the definitions are given by Howe (1993). The order of the terminology has been designed to form a logical rather than alphabetical order since the terms build on each other. The terminology has been adhered to as near as possible throughout the text and some of the definitions are expanded upon in the main document.

- file — An electronic entity that contains binary data. This data may represent a collection of characters such as letters and digits or it may form a program that can be executed by a computer.
- source (code) — A particular type of file that can be compiled into an executable program.
- document — A specific type of file that is structured in XML.
- (document) fragment — This is a file that is structured in XML although it may not in itself form a complete document.
- web page — A file that is presented by a browser in a human readable form. These files are typically in HTML format.
- transformation — A file that is used to manipulate a document. A transformation generally refers to an XSL file.
- style-sheet — A file that is used to define the visual format of another document. A transformation is a particular type of style-sheet. Cascading Style Sheets (CSS) are another form that is used to define the style of a HTML file.
- schema — A representation that describes the structure of a family of related concepts. In terms of XML, a schema is a document that provides the content model of a family of XML documents. Similarly, the IFC Model is a specific schema to describe buildings.

- domain — A particular subject area that is described by functions, objects, data, requirements, relationships and variations. Domains have a limited scope specific to a subject field or interest group e.g. structural design, architecture or fire engineering.
- entity — Something that has a particular separate and distinct existence and objective or conceptual reality. In this thesis an entity is often used in the context of the IFC Model and the contents of a domain.
- class — A class is defined in object-oriented programming by set of state variables and behavioural methods associated with the class. Classes can be related in a hierarchy where one class is a special case (the "subclass") of another (its "superclass") or several other classes. This thesis uses C++ classes in its implementation of the IFC Model entities.
- object — An instance of a class that has its own values for the class variables and can respond to the methods defined by the class.
- (computer) program — An executable version of a piece of source code usually generated by a compiler.
- (computer) software — A combination of one or more programs that together can be run on a computer.
- (software) package — A package consists of generally one or more software items plus supporting files, libraries, documentation etc. This may sometimes be referred to as an 'application'.
- (computer) platform — A combination of computer hardware and operating system software. For example, a desktop PC with an Intel processor running **Microsoft Windows XP** might be described as a 'Windows platform'.
- parser — Software that can be used to tokenise and understand parts of a document.
- browser — Software that is used to present web pages.

- simulation tool — Something that simulates a process through the use of computer software. The simulated process is often a physical system but not necessarily so.
- fire simulation tool — A computer program (or package) that simulates some process related to fire engineering. The tool may be designed to simulate one or more processes such as combustion dynamics, heat transfer, fluid flow, structural response, people movement etc.
- model — A mathematical or relational representation of a product or process. In fire engineering the term "fire model" is often used to describe what is referred to in thesis as a fire simulation tool.

Chapter 1:
INTRODUCTION

1.1 Background

Information technology coupled with the rise in the use of personal computers has developed rapidly over the past twenty-five years. In that time the capabilities of personal computers and their ability to communicate over the internet have increased at an astounding rate and what was almost unimaginable in the 1980's is old news in 2005. Early developments in the personal computers led to much time and effort spent reworking the technology to make use of new advances. Computer systems and their file formats were rarely compatible with each other leading to an inevitable loss of data and inefficiency. As the technology matures, solutions to the loss of data or the need to rework data are becoming important considerations.

The increasing power and availability of desktop computers has led to their introduction in almost all facets of modern society. The construction industry is no different and there is an ever-increasing use of computer software to the whole building life cycle. Examples include the initial mapping of the site using Geographical Information Systems (GIS), the use of Computer Aided Design (CAD) packages, specialist tools for simulating performance of the structure and its living environment, construction management tools and systems incorporated into the final structure that monitor the day-to-day operations of the building.

Ideally each of these diverse software packages should to be able to exchange information. To the end user, this exchange would be a seamless process in which there is no loss of information and no need to manually intervene (for example by having to modify the format of physical structure of the file). However, the construction industry is a diverse group, ranging from individuals to large multinational corporations, involved in an extremely diverse number of activities; architects, suppliers, engineers, contractors, managers etc. This diversity has meant that compared with other industries, for example the aerospace and automotive industries (Ungerer, 2003); there are significant barriers to the integration of software tools and the exchange of information between them.

The benefits of being able to exchange building information electronically are seen as including more efficient transfer of data between design and analysis tools across different disciplines and throughout the life-cycle of the building; keeping the building description, analysis methods and drawings consistent; the availability of different 2D and 3D (and possibly 4D, if time is included) views of a building to be generated from a single description; the capability to export the building description for further manipulation. All of these benefits have the potential to result in efficiency gains that will result in more cost-effective buildings.

Fire engineering is a small but integral part of the construction industry, typically 2-6% of the total construction cost of a building. Computer software can be used to simulate the dynamics of fire, the response of structures and the movement of people (Olenick and Carpenter, 2003) and are broadly referred to as 'fire simulation software (tools)' in this thesis. The ability to exchange electronic information between fire simulation software and software used by other parts of the construction industry can assist in achieving some of the efficiency gains described above. However, developments in integrated data exchange are likely to be led by the larger players in the industry and it is important that the fire engineering discipline is aware of wider issues so that it does not get left behind.

1.2 Objectives

The primary objective of this thesis is to examine the practicability of implementing electronic data exchange in fire engineering using emerging industry standard technologies. Fire engineering can often include aspects of hazard identification, fire growth and smoke movement simulation, egress assessment, structural fire performance, sprinkler and/or fire detection system design, smoke management system specification. Fire engineers also need to be conversant with regulatory requirements, the application of standardisation documents, accepted industry practice and needs of the emergency response services.

This thesis focuses primarily on aspects of fire growth data and computer software that simulate fire growth and smoke movement. Specifically the thesis aims to demonstrate the feasibility of such electronic exchange by integrating recent advances

in construction information technology with commonly available fire simulation tools with outcomes of improved efficiency and accuracy for fire engineering designers and all others with whom they interact.

A more general objective of this thesis is to raise the awareness of a standard building model (known as the "IFC Model") amongst the fire engineering community so that developers of fire simulation software tools can more closely integrate their work with this model. Likewise, this thesis attempts to demonstrate to the authors of the standard building model the needs of the fire engineering community and in particular the requirements of the developers of fire simulation software.

1.3 Thesis structure

A summary of each chapter in this thesis is given below. In general terms, this thesis starts from a generic view of information and its exchange, then focuses on this problem as it relates to the construction industry as a whole and more specifically to the fire engineering field. It then goes on to examine specific technologies for the exchange of fire-related information and demonstrates the feasibility of creating buildings in commercially available CAD packages which can easily be transferred to commonly used fire simulation software using a standard building model.

The thesis includes six chapters that have been published or submitted to two peer-reviewed journals, three peer-reviewed conference proceedings and a professional journal. Although each paper addressed a different aspect of the topic to a different audience there is necessarily some repetition of content amongst the papers. Several references appear in two or more of the papers and some of the content of one paper is similar to the content of one or more of the others. However, each paper presents a different component of the overall research outcomes to form a complete body of work. Chapters published as papers also often include additional work or material that was omitted from the original submission and this is given at the end of the relevant chapter.

Those chapters which were initially published as papers have been re-formatted for the thesis so as to present a consistent style. References in each paper have been

amalgamated and appear in Chapter 13. Cross-references to chapters in this thesis have been used to replace references that were originally citations to the papers that have now been used as part of this thesis. In one or two instances the content of the original papers have been slightly modified to make the chapters flow better or to correct minor errors noted after publication. Spelling in this thesis generally conforms to normal New Zealand English but since some of the papers were printed in US publications there are instances of US English, particularly in diagrams. The following paragraphs give a summary of the content of each chapter that forms the thesis, with reference to their original place of publication where appropriate.

Chapter 2 describes the role of information technology in construction and fire engineering. The chapter briefly reviews the progress to date in the area of integrated data exchange and looks at what may be possible in the future. Chapter 3 gives some general background to the specific web-based mark-up technologies discussed in several of the remaining chapters. Central to this discussion is XML (a superset of HTML used to display web pages) which is extensively used during the development of this thesis. Chapter 4 (Spearpoint, 2003a) introduces the concept of a building product model and specifically the IFC Model. The chapter describes the technology that was available at the time then briefly examines what the future might hold and in particular how these technologies might impact the fire engineering profession. Chapter 5 (Spearpoint, 2001) describes the development of a database of heat release rate information. The database can be used as a stand-alone tool but was also developed with the wider goals of this thesis in mind. Chapter 6 describes how to interface a database of heat release rate information using web-based mark-up technologies and other software tools. The interfaces provide users with ways in which they can obtain information from and on a database. The chapter then details techniques to extract and convert database records using several different methods. Chapter 7 (Spearpoint, 2003b) gives some background to fire engineering design and the New Zealand regulatory environment. It then examines the content of the earlier IFC 2x Model in relation to fire engineering and shows how the database of heat release rate information can be used to enhance the IFC Model.

Following on from Chapter 7, Chapter 8 (Spearpoint, 2003c) demonstrates the translation of XML format IFC files into a form that can be used as input to two

commonly available zone fire simulation tools. Chapter 9 reviews the fire-related properties in IFC 2x2 and compares this later release with the earlier IFC 2x version. IFC 2x2 gives significantly greater support to fire engineering but still has aspects that could be improved. Chapter 10 (Spearpoint, 2005a) then illustrates how the fire engineering support provided in IFC 2x2 can be mapped to a commonly available fire simulation software tool. Finally, Chapter 11 (Spearpoint, 2005b) demonstrates the exchange of IFC 2x2 files between an existing CAD package and the fire simulation software tool.

An overall set of conclusions and recommendations are given in Chapter 12 and Chapter 13 gives a complete set of references used in this thesis. Several Appendices are included that provide unpublished supplementary information and details of the information storage schema and transformation methods.

1.4 Software

Several software packages and tools specifically related to XML technologies were used during this study. For viewing web pages Microsoft **Internet Explorer (IE)** versions 5.0 to 6.0 were used. The web browser supports XML technology through the **MSXML** parser described below. **XMLSpy** versions 3.5 to 4.2 (Kim, 2003) were used extensively to create and edit the XML, XSL, XSD and HTML documents.

The Microsoft XML parser was used by both **IE** and **XMLSpy**. During the progress of this study the parser went through several version changes partly to keep in line with the XML specifications. Most of the initial development work used **MSXML 3.0** which complied with the version 1.0 specification for XML (W3C, 2000) and subsequent work used **MSXML 4.0**. Similar XML parsers are available for other computer hardware and operating systems.

Other more general software was also employed during the progress of this study. Microsoft **Visio Professional 2002** was used to create many of the diagrams used in this thesis. It was also used to generate and view building descriptions as described in Chapter 8. Graphisoft's **ArchiCAD** version 8.1 (Graphisoft, 2005) was used to create

CAD representations of specific buildings and export them in IFC format. Further details are discussed in Chapter 11.

Many of the web pages written in the study were created using Microsoft **FrontPage 2002**. Microsoft **Visual Basic 6.0**, Microsoft **Visual C++ 6.0** and later Microsoft **Visual Studio .NET** (Simon and Schmidt, 2002) were used to create the software developed during the study. Finally the **BRANZFIRE** (Wade, 2002) and **CFAST** (Jones et al., 2000) zone fire simulation tools, briefly described in Chapter 8, Chapter 10 and Chapter 11, were used to demonstrate the ability to exchange IFC files and to integrate the database of heat release rate data developed in Chapter 5.

Chapter 2:
INFORMATION TECHNOLOGY IN CONSTRUCTION

2.1 Data and information

Before examining the specific needs of the data exchange within the construction industry and specifically in fire engineering, it is necessary to consider what is the difference between 'data' and 'information' (and ultimately 'knowledge' and 'wisdom'). Data has been described as follows (Howe, 1993):

> **"data** <data, data processing, jargon> /day't*/ (Or "raw data") Numbers, characters, images, or other method of recording, in a form which can be assessed by a human or (especially) input into a computer, stored and processed there, or transmitted on some digital channel. Computers nearly always represent data in binary.
>
> Data on its own has no meaning, only when interpreted by some kind of data processing system does it take on meaning and become information.
>
> People or computers can find patterns in data to perceive information, and information can be used to enhance knowledge. Since knowledge is prerequisite to wisdom, we always want more data and information."
> [*underlined words have separate definitions*]

There are many definitions for information that include artificial and biological systems. One definition of information is (WordNet, 1997):

> **"information** n 1: a message received and understood that reduces the recipient's uncertainty [syn: info] 2: a collection of facts from which conclusions may be drawn; "statistical data" [syn: data] 3: knowledge acquired through study or experience or instruction 4: (communication theory) a numerical measure of the uncertainty of an outcome; "the signal contained thousands of bits of information" [syn: selective information, entropy] 5: formal accusation of a crime"

In this thesis the 'information domain' is limited to computer systems and the information exchange between them. One thing is clear though, there is little point

having data if it cannot convey information. Consider a string of numbers such as 24.0, 22.3, 18.6 etc. By themselves they only constitute data and it is not possible to attach any particular meaning to them apart from knowing that they are numbers. Being told that they are daily maximum temperature readings transforms the data into information. However, it would be necessary to know much more before the data might be used to gain knowledge. For example, where were the readings taken? (geographic location and physical position), what instrument was used? how accurate was it?, are the data average readings or taken at some specified time?, what units are the data measured in? With this information brings us knowledge of the environment we are measuring and this will hopefully lead us to make wise use of the knowledge. We can now begin to understand the particular environment and thus make decisions using our intelligence and the acquired knowledge.

Clearly then, in order to gain wisdom one needs knowledge. This knowledge is derived from information that, in turn, is obtained from data. Figure 2-1 shows one way in which this description can be viewed diagrammatically.

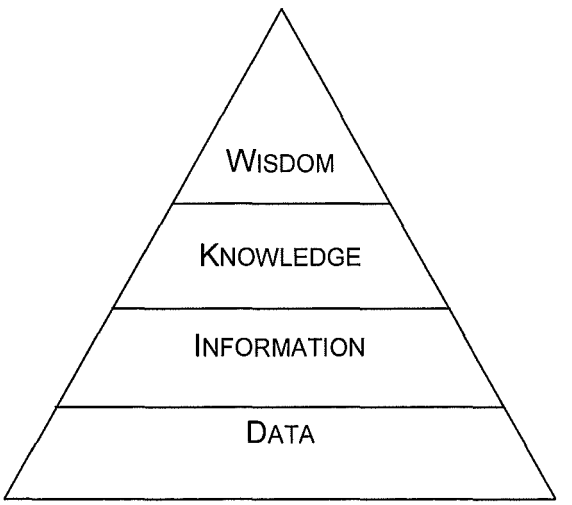


Figure 2-1. The data-wisdom pyramid.

Similarly, the connections between these concepts can be written as (adapted from Lietaer, 2001):

Data is undigested observations without context.

Information is systematically organised data according to some system aimed at making it retrievable and hopefully useful.

Knowledge is information that has been internalized, integrated into everything else gained from experience and study, and is therefore available as a basis for action. An increasingly important form of knowledge is learning how to find information that is useful

Wisdom adds depth, perspective and meaning to knowledge by integrating ways of knowing other than by logic and analysis, such as by intuition, or the intelligence and compassion from the heart. Wisdom is by definition multi-dimensional, crossing the boundaries between different fields and ways to knowledge. It is the ultimate synthesis which cannot be forced on or taught to someone else.

2.2 Information model

In order to interpret information, it needs to be presented in a way that is usable to a client. The 'client' may be a human being or a computer. Thus we might consider information can be broken down into the following 'information model':

- Media. This is the physical aspects of information storage and transfer e.g. a printed book, the internet.
- Language. This is the agreed form of transmitting the information from the author to the client using letters, words and grammar. The letters are the actual characters that form the language e.g. the Latin characters or Greek characters. These characters are placed into groups to form words and the words are placed in a certain order to meet the grammar requirements of the language e.g. French, English, and FORTRAN. These rules are not necessarily the same in all languages, for example, in some languages a single character is a word.
- Semantics. The semantics provide meaning to the words through their relationship to other words. In a construction context, the relationships between words may be used to describe the properties of an item and how is that related to other items. For example, in fire engineering, the word 'vent'

could mean either a window or a fan and thus the 'size' of a vent is meaningless until we know type of vent is being considered. The size of a window might mean one or more of its physical dimensions whereas the size of a fan might mean its rated power output.

- Structure. The document structure organises the information where the same information can be structured in several different ways. An example of structure is the use of paragraphs, sections and chapters in a paper-based document. Structure facilitates the finding and retrieving of information as well as understanding.
- Presentation. Finally information can be presented using a range of styles that might include different fonts, colours etc. or headings, layout etc.

This research is delivered as a thesis that has a specific 'structure' of chapters and headings written in the English 'language' which uses the Latin character set. The 'medium' is a printed document (or an electronic equivalent) and is 'presented' in a range of fonts and colours. Within the thesis, a specific set of 'semantics' are described.

2.3 Sharing and integration

As already mentioned, the exchange of information is a major issue where computers are used to store and manipulate data. Developers often create software in isolation and have to accommodate constraints of the environment they work in (knowledge limitations, available database tools, existing programming languages etc.). The data is stored in the 'information model' that is most convenient for them. Without a standard system for the content and transfer of data between tools, conversion processes are often necessary. Each conversion process may 'devalue' the data as the content of the data has to match the lowest common format. An example is converting a word processor file to a text file. The conversion to the text file loses formatting styles such as bold or italic text. Even if the 'same' file is then imported back into the word processor, the formatting cannot be recovered. We might call this 'Chinese whispers' data exchange – each exchange of data potentially 'devalues' the content.

Until recently the exchange of data between applications has been very limited, much like the scenario shown in Figure 2-2. In many cases applications have been unable to share data in any simple way. Data has had to be manipulated either manually, or via some separate conversion tool. Alternatively data had to be re-entered manually which is time consuming and can lead to errors.

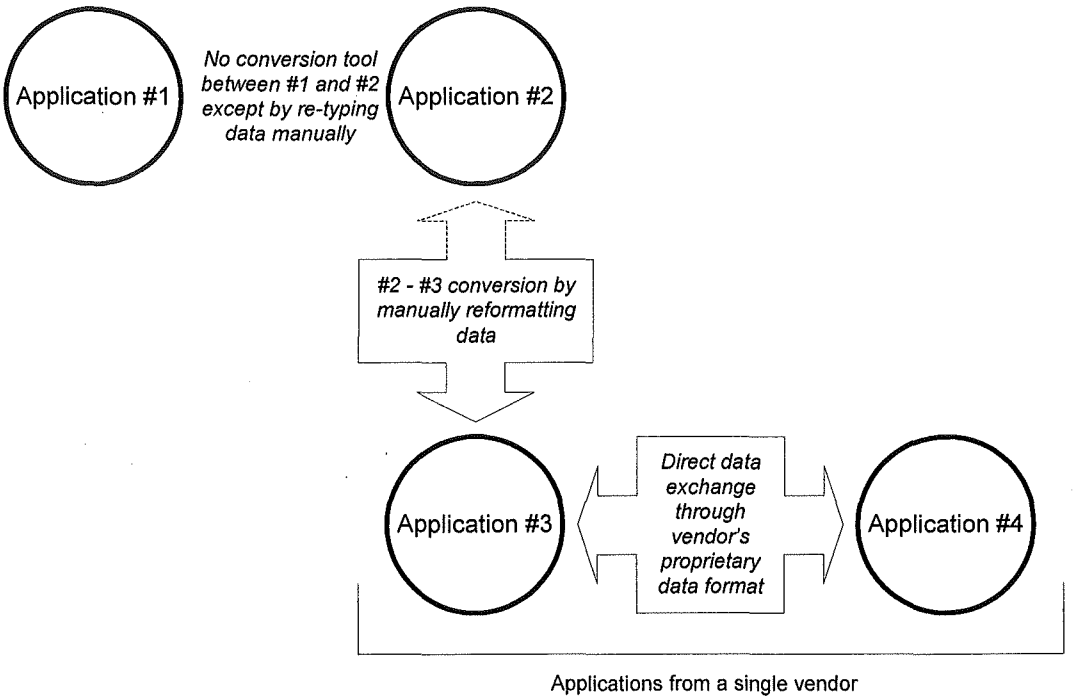


Figure 2-2. Past situation: the exchange of data between applications was limited.

In some cases, applications developed by a particular vendor may have had some proprietary data exchange tools (as shown in Figure 2-2 between Application #3 and #4). However the data format may not have been publicly available, making the development of additional exchange tools unlikely.

The current situation, at least in fire engineering, is probably closer to the scenario shown in Figure 2-3. Data can be transferred between applications through individual conversion tools. Each conversion tool is separately maintained and this approach may still lead to loss of detail in the data. In Figure 2-3 the exchange between Application #3 and #4 is essentially the same as in Figure 2-2.

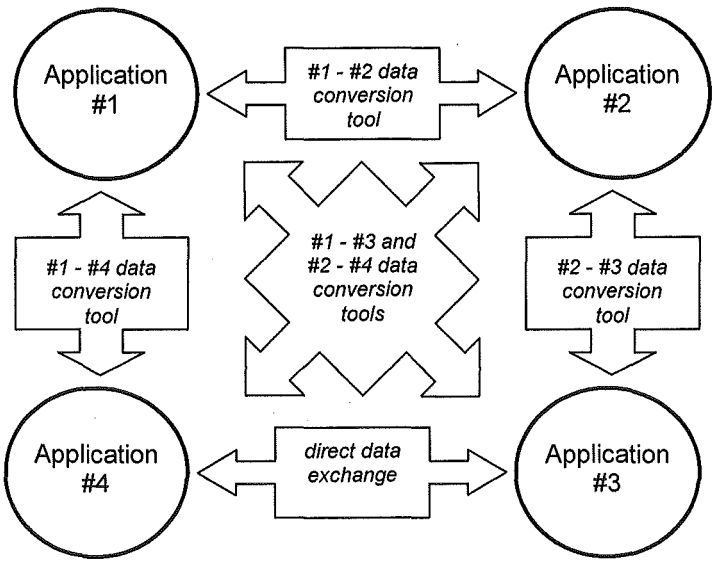


Figure 2-3. Present or near-future situation: applications can move data between individuals through separate conversion processes.

As additional applications are introduced, additional conversion tools may be required and if an application is updated, the corresponding conversion tools may also need updating.

Figure 2-4 shows what may be in store in the not-too distant future assuming that construction engineering continues to progress to an integrated data modelling environment. Instead of having numerous data formats and thus conversion tools, a common data format is specified. Applications exchange their data through a 'central' database (or repository) management tool. This repository may be on the local computer or a remote connection through a Local Area Network (LAN) or Wide Area Network (WAN).

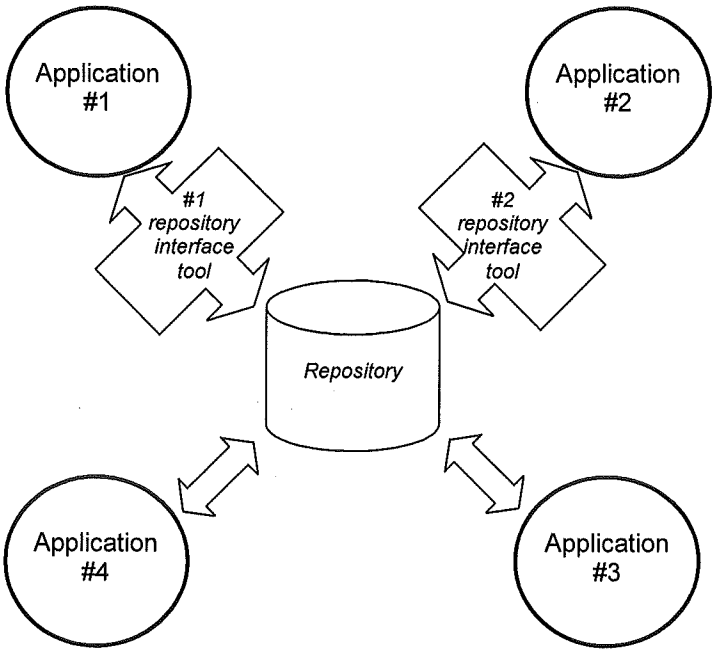


Figure 2-4. Future situation: applications exchange data through a 'central' repository. Some older applications may still require a separate repository interface tool.

This approach eliminates the need for numerous conversion tools. It also means that if one of the applications is updated, only the interface to the repository need be updated. If a new application is introduced to the process then only an additional interface is needed to and from the repository. Some legacy applications will still require a conversion tool to interface to the repository particularly where access to the original application source code is difficult to obtain or where rewriting the application to directly interface to the repository is not feasible.

Eventually we might envisage a situation similar to that shown in Figure 2-5 where each application directly interfaces to the repository. This scenario may be far into the future and many not ever exist where legacy applications that are no longer being developed are still in common use.

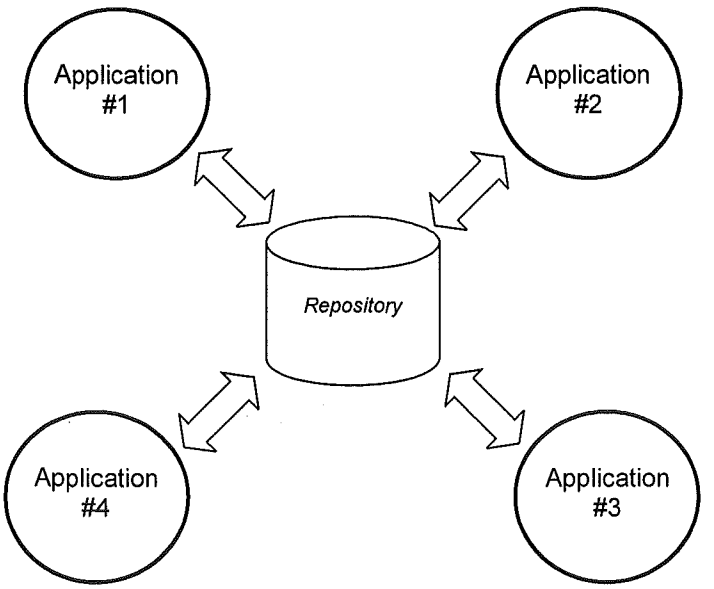


Figure 2-5. Far-future situation: all applications exchange data through a 'central' repository.

Although in Figure 2-4 the eventual development of a repository based information exchange is described, there are alternative approaches to interoperability. For example, instead of transferring information to the repository, information could be communicated directly between applications, but each transfer is verified against a prescribed schema. The verification process ensures that the file conforms to a given set of rules. The rules can be used to check that the file is correctly formatted, that information is present where appropriate, and that information is in a correct form. The repository is no longer necessary if an agreed schema is determined and every transfer is verified against that schema.

The model suggested in Figure 2-4 and Figure 2-5 also assume that it is possible to store all of the information required to describe a building across all disciplines and throughout its life can be stored in a single standard representation. However, as noted in Section 1.1, the construction industry is a diverse group involved in an extremely wide range of activities. It may be virtually impossible to create a single representation for all these groups and activities to a sufficient level of detail that every piece of information required for all domains is encompassed in a whole project representation. Instead it may be more practical to create a central project

representation that contains the most common elements and allow domain specific information to be held outside the central representation. Mechanisms in which to store and obtain domain specific information may then be required to support such an approach.

2.4 Information technology in the construction industry

Structures, such as buildings or ships, consist of a collection of components or 'entities'. These entities may be physical objects such as doors, windows, walls etc. or more conceptual entities such as a space. The current generation of CAD tools are not able to describe the performance or properties of the entity (and its component parts). A common feature in all CAD packages is the ability to transfer files in Data Exchange Format (DXF), the de facto format established by the developers of AutoCad and adopted by many other system developers. However DXF files were originally designed to represent entities as a collection of points and arcs. They are less able to store additional information such as density, thermal conductivity etc in a standardised way. The Open Design Alliance (2003) notes that the DXF specification is not an open standard format that can be used by all users. The specification is proprietary and in some instances includes encrypted data. The DXF specification has been imprecise and prior to Release 14, information could be legitimately not provided in a file and still be considered valid. The DXF specification is loosely defined which forces software to be very forgiving about the quality of the DXF files.

There are many developments at an international level, both within the construction industry and in commerce in general, for standards that describe methods for storing, transmitting and manipulating meaningful 'domain' data in an open environment and some of these are discussed in this thesis in later chapters. Without these standards, industry will continue to be at a disadvantage because of the lack of integration between its various proprietary systems. Projects will continue to require error-prone manual interpretation and re-entry of information which is a labour-intensive process.

The format of the data corresponding to a specified domain is specified as a schema. Data that describes a particular building structure conforms to this schema. The schema describes the type of entities, the properties of those entities and the inter-

relationship between entities. It is often not the intention that the schema describes every aspect of a building down to the detail required by every engineering discipline involved in the construction process. This would make the schema too unwieldy. The schema is more likely designed to describe the most common parameters that may be required across several disciplines. Individual design tools may still need additional data to be entered by some other route (e.g. a separate database of specialised material properties).

The development of IT in the construction industry, like all others, is a rapidly changing environment. Therefore some of the ideas presented in this thesis are likely to be superseded by new ideas and applications. It is recognised that specifications for data description and exchange will evolve over time. The current industry-wide approach appears to be that it is preferable to evolve this technology in stages rather than conducting an extensive study that may require many years before anything becomes available for use by the industry. Software tools, like the infrastructure they design, have their own lifecycle.

2.5 Developments in construction IT

This section will only provide a short summary of the information technology developments in the construction industry. There have been a number of government, academic and commercial organisations working in this field over the past decade or so. The work has encompassed many aspects of the building life-cycle with varying degrees of success. Various commonly used terms have been used to encompass this work including Integrated Data Modelling, Computer-Integrated Construction (CIC) and Building Information Modelling (BIM).

In 1997 the construction industry identified "the fully automated, one time data entry, seamless integration of project life cycle work processes as the most significant currently identified trend and predicts it will revolutionise the industry" (Cassidy, 1999). Computer-integrated construction is an outcome of computer-integrated manufacturing and has been defined as "a better use of electronic computers to integrate the management, planning, design, construction and operation of constructed facilities" (Jung and Gibson, 1999). The above definition indicates the enormous

scope of CIC projects. For example, significant work is on-going with regard to the electronic processing of construction project documentation (for example, Caldas et al. (2002), but there are many others). The shortcomings of traditional CAD systems and the benefits of object modelling have been discussed in detail by Finch (2000). He also discusses the roles of STEP and XML (see Section 3.3) in using the Internet to share construction information.

This author first identified limitations of CAD packages during discussions relating to the development of fire modelling software tools (Spearpoint, 1992a) and a research project in which the use of the **Superscape** Virtual Reality (VR) system was being proposed as a means of conducting fire risk assessments (Spearpoint and Smithies, 1993). Some progress was made in using the VR system (Spearpoint, 1994a) but ultimately the project never reached full fruition and an alternative approach was taken (Fraser-Mitchell, 1994). However, the VR system was used to recreate a then commonly used fire model (Spearpoint, 1994b) and was also later found useful in the reconstruction of actual fire events (Spearpoint and Shipp, 1997).

The protocol called STEP (STandard for the Exchange of Product model data) is the ISO standard for product modelling (ISO, 2002). The goal of the standard is to provide a complete, unambiguous, computer-readable definition of the physical and functional characteristics of a product throughout its life-cycle. STEP is designed to allow for the sharing technical data about products such as automobiles, buildings, consumer electronics, planes, ships and trains. EXPRESS (ISO, 1994) is the information modelling language used to define the STEP standard and EXPRESS-G (ISO, 1994) is a graphical version of EXPRESS. More details regarding STEP, EXPRESS and EXPRESS-G are provided later in this thesis (see Chapter 11).

In Europe, a consortium of research institutions began work on the COMBINE I model in 1990 (Augenbroe, 1994) with the aim of integrating building design software tools. The COMBINE I model was later extended into the COMBINE II model (Lewis and Kenny, 1994). The COMBINE I and COMBINE II projects developed STEP schemas to describe buildings and to use these schemas within applications to improve building energy efficiency. These COMBINE schemas were further enhanced for the wider construction community in the BREED model (Cheng,

1995). The current author was involved in the BREED model development and in particular examined techniques to interface the BREED model with the Superscape VR package and also with fire-related software tools (Spearpoint, 1995a). The BREED model was successfully interfaced to Superscape (Spearpoint, 1995b; Spearpoint, 1995c; Spearpoint, 1996) and also progress was made on interfacing the BREED model to commonly available fire simulation tools (Spearpoint, 1995d; Chitty and Spearpoint 1996) but funding was not available to continue the development any further.

However, some of the outcomes from the COMBINE and BREED projects have resurfaced with a new interest in the sharing of electronic information in the construction industry. A group of research and commercial organisations have formed a group known as the International Alliance for Interoperability, IAI (IAI, 2004a). The group recognises the technical and commercial benefit of being able to share construction related information. Currently the IAI is developing a standard set of classes (the Industry Foundation Classes or IFC Model) that specify how entities that occur in buildings should be represented electronically. As with the STEP standard, the IFC model notation uses EXPRESS-G but the growth in web-based languages have been adopted (Liebich, 2000) to further develop the IFC Model notation.

2.6 IT in fire engineering

2.6.1 General usage

Fire engineers use IT for many purposes. Similar to almost every modern industry or business, fire engineers use IT for document exchange and storage. This includes word-processing, CAD drawings, email etc. In many cases these documents are stored in a haphazard manner and may be difficult to organise or to retrieve specific information for them. The developments in electronic document management attempt to resolve these limitations, specifically for construction projects where fire protection may be identified as a specific division of the project (for example, Caldas et al., 2002).

2.6.2 Fire simulation software

Fire engineers also use computers for design work, for both CAD plans and in particular for simulation modelling. Olenick and Carpenter (2003) have identified a wide range of computer simulation software available to fire engineers. The tools can be used for various fire engineering related tasks such as predicting fire and smoke spread, determining the performance of structural elements under fire conditions and the analysis of people movement in buildings or other premises. The software tools vary in the extent of the fire hazard scenario represented and subsequently the complexity of the input requirements and the sophistication of the output capabilities.

Olenick and Carpenter (2003) categorise the software tools (or 'models' as they refer to them by) as zone models (as discussed in Section 8.3), field (or Computational Fluid Dynamics, CFD) models, detector response models, egress models, fire endurance (structural) models and miscellaneous models. Miscellaneous models include air-flow movement models, risk analysis tools and suites of sub-models that may be found in one or more of the models included in their main categories.

Thus the term 'fire simulation software' is used in this thesis although it can be seen from the list of functions such software is intended to carry out that a fire is not necessarily simulated in its entirety or even at all in some cases. Often when a structural performance is being undertaken the fire is represented as a time-temperature curve rather than being simulated directly. In some of the egress analysis software, the fire is not explicitly included at all because it may be sufficient to assume that occupants are remote from any effects of a fire.

2.6.3 Fire simulation software parameters

Section 2.6.2 above illustrates the wide range of fire simulation software that is available and the development of procedures to exchange BIM data with all these tools would be a significant task. Instead, the focus of this work is the interfacing of the IFC Model with those fire simulation software tools that can be used to analyse the dynamics of fire and smoke. Even that task requires considerable effort and so this work specifically examines those software tools that use the zone modelling approach to fire and smoke dynamics. In order to get some notion of what is required by this

type of fire simulation software it is useful to examine what kinds of parameters are needed by the software tools in question.

Appendix A.1 summarises the various input parameters of two fire simulation software tools, namely **FPEtool** (Deal, 1995) and **CFAST** (Jones et al., 2000), reviewed by Spearpoint (1992b). The parameters are broken into related groups that consider the environmental conditions; space geometry, topology and boundary conditions; materials; fire safety equipment and mechanical services. Each tool has a certain number of common parameters and other parameters that are unique to that tool. Not all parameters are required to conduct a simulation and in some cases a suitable default value is provided by the tool if no other information is available. In addition to these parameters, each tool requires a specification of the burning item and this is described in more detail in Section 5.8.

It can be concluded from this earlier review that information required by fire simulation software includes the geometry and topology of the spaces (e.g. room dimensions and openings), the properties of the space boundaries (e.g. wall thickness and material properties). There may also be specialised non-fire engineering information required such as the ventilation flows and temperatures in spaces prior to a fire. Finally, there may be specific fire-related information required such as the burning characteristics of fuels, the thermo-physical properties of materials and the features associated with fire safety equipment.

Chapter 8 and Chapter 10 build on the findings of this review and previous fire simulation analyses conducted by the author (Spearpoint, 1992c; Spearpoint, 1992d; Spearpoint, 1993b; Spearpoint, 1998; Spearpoint et al., 1999). In particular Chapter 10 examines what information is available in the IFC Model that can be used to transfer to the **BRANZFIRE** fire simulation tool.

2.6.4 Integration of fire engineering packages

Although the developments in the exchange of electronic information in the construction industry have been discussed previously, in the fire engineering field there appears to be almost no work currently taking place in the open literature. Previous work by Mowrer (1987) presented a methodology for the use of CAD

software as a means of providing an object-oriented description of a building and the provision of associated attributes to fire simulation tools. The implementation of a prototype CAD-based building fire safety analysis system was described. Independently, this author highlighted in previous research the need to be able to share electronic data between fire programs (Spearpoint, 1992a) and some initial progress was made.

In the work being undertaken by the IAI, only one specifically fire-related project has been identified and this is to investigate the regulatory requirements for emergency escape (IAI, 2004b). The project includes two stages in which the first stage assesses the compartmentation of building for the purpose of means of escape in case of fire. The second stage establishes escape routes from all parts of the building to the final exits and then checks for compliance with the regulatory requirements. However, at the time of writing, this project appears to be on hold due to resource constraints.

Fire-engineering computer packages and related software can be run on a range of platforms from desktop personal computers running Windows-type operating systems to more powerful workstations operating in Unix-type environments. Clearly any developments in electronic information exchange must not be reliant on any one particular type of hardware, operating system or specific package.

Currently almost all of the information required by a computer fire simulation tool has to be entered manually. For even relatively simple simulation tools this data entry can involve up to 100 variables (Spearpoint, 1992b) and the data requirements can increase significantly when considering the modelling of complex scenarios using some of the more powerful simulation tools that are available. Manual entry is not only time-consuming but potentially leads to mistakes. One of the perceived barriers identified to more widespread use of fire simulation tools has been the lack of integration with CAD and thermal software tools (Bloomfield, 1994) which would address some of these problems.

Researchers have investigated techniques to automate the recognition of CAD drawings (Berkhahn and Esch, 2003) using a computer to transfer such results to a fire simulation tool (Frost et al., 2001) with some success. The **Simulex** evacuation

simulation tool (Thompson and Marchant, 1995) allows the direct import of CAD data through DXF files. These methods often require that the original files be 'cleaned' of data that would otherwise be incorrectly interpreted by the target software. The limitations of the DXF format also mean that only basic geometry can be derived from the files. The procedure can also be error-prone and reasonably time-consuming. For example the recognition process described by Frost et al. (2001) requires the following seven steps to transfer a DXF file to the **SMARTFIRE** program:

Step 1: Original DXF file with/without any "cleaning up"
Step 2: Define storey after partial "cleaning up"
Step 3: Generate rooms using ScanLine Fill Approach (SLFA) algorithm
Step 4: Manually add rooms not recognised
Step 5: Add doors and windows (at present manually but can be automated)
Step 6: Create the fire scenario of interest
Step 7: Export to **SMARTFIRE**

2.6.5 Current use of the web

From a fire engineering perspective the web already has a number of uses. Surveys of web resources (Jason, 1996; Lundin, 2001) have identified databases of references, collections of research material, software packages, journals and email newsgroups. In the case of research material, there are several organisations that publish their research online as PDF files. However, very few organisations offer their data in an accessible electronic format. There may be several reasons for this including commercial confidentiality, copyright concerns but also because there is no widely accepted format for the storage and transmission of the data.

Chapter 3:
WEB-BASED MARK-UP LANGUAGES

3.1 Introduction

This chapter discusses various aspects of web-based mark-up languages. In particular the XML language which forms much of the basis of this study. It is not within the scope of this thesis to go into great detail regarding the technology and further information is available elsewhere (Light, 1997; Harold, 1998).

In this thesis the XML language is used in Chapter 5 to create a database of rate of heat release data in a form that can be used by web pages, by a specifically written client application or by fire models developed by third parties. Furthermore, XML is one of the technologies used by the IFC Model to exchange building information between applications as will be demonstrated in Chapter 8.

3.2 XML related technologies

Before discussing the details of XML a brief examination some of the technologies than have preceded it are presented. Firstly, Standard Generalized Markup Language (SGML) (ISO, 1988) is an international standard for semantic tagging of documents and is particularly intended for computer cataloguing and indexing. SGML describes the relationship between the content and the structure of a document. This allows document-based information to be shared by applications and computer platforms in an open, vendor-neutral format. SGML is a collection of 'nodes' organised into a 'document tree'. Nodes can be attributes, elements, comments, markup declaration or processing instructions. Tokens, referred to as 'tags' are used to represent the beginning or end of an element. SGML is very complex and is suited to large quantities of highly structured data and, as such, is popular in the defence industry.

The second technology to mention is Hyper-Text Markup Language (HTML). It is aimed at presenting documents to a human rather than being understood by other software. As described by Light (1997), HTML is an application of SGML that is very successful as a means of creating formatted documents such as web pages although it has severe limitations. The tags available in HTML are predefined so it is not possible for someone to simply add a new tag and for that be understood by HTML compatible software. This restricts the use of HTML to specific tasks where there is not the requirement to be able to develop tags for new applications.

Finally, Cascading Style Sheets (CSS) were originally designed to tell a browser how to format individual tags in an HTML document. CSS defines properties such as font characteristics, paragraph formats etc.

3.3 XML

3.3.1 The language

Extensible Markup Language (XML) is a meta-markup language which provides a format for describing data in a structured manner. The structured language facilitates precise declarations of content and more meaningful search results across different computer platforms. XML is a subset of SGML that is optimised for delivery over the web.

The original work on XML began in 1996 and related technologies were developed during 1997. The XML language is being managed by an international group called the World Wide Web Consortium (W3C) (World Wide Web Consortium, 2001), who are ensuring that structured data will be uniform and independent of applications or vendors. The group aims to develop interoperable technologies (specifications, guidelines, software, and tools) to lead the web to its full potential as a forum for information, commerce, communication, and collective understanding. Tools to manipulate and view XML files are already freely available. For example, files can be viewed using Microsoft's **Internet Explorer**.

Although XML and HTML are both related to SGML, it is important to distinguish between their differences. HTML is an application of SGML with the specific purpose of online display of webpages. XML operates at a much more general level and can be used for a limitless variety of applications (Light, 1997).

3.3.2 Documents

Like SGML, an XML document consists of a collection of nodes. There are several types of nodes including elements and attributes. Element contents are bracketed by a pair of tags e.g. <tag>Content</tag> except where the tag has no content e.g. <tag/>.

XML has no predefined tags or limitations on their number. It provides a data standard that can store the content, semantics and schema of a data-set.

A simple XML document is shown in Figure 3-1. An XML document must have a single unique top element called the 'document element' (although the tag does not have to be '<document>'). Elements can have any number of sub-elements nested to any depth and any amount of text can be included. An example of an element might be the details of a reference that has sub-elements of the title and the name of the authors as shown in Figure 3-1.

```
1: <?xml version="1.0" encoding="UTF-8"?>
2: <document>
3:     <reference type="Conference proceeding">
4:         <title>Advances in Assessment Methods for Fire Safety</title>
5:         <authors>Williamson R B, Dembsey N A</authors>
6:     </reference>
7: </document>
```

Figure 3-1. A simple XML document.

Additional information can be added to an element by using attributes. Attributes have a name and a value. Values are optional and an attribute can exist as a name only. An example of an attribute would be specifying different types of reference source: conference proceeding, report, journal paper etc also illustrated in Figure 3-1. Thus, a search of the XML document can specify which types of reference source should be accessed, in a manner similar to a standard database.

3.3.3 Schemas

The 'schema' describes the structure of an XML document in terms of the relationship between elements and attributes and the types of data that can be stored within them. A specific XML document can be automatically checked against a schema for conformity. Schemas for XML documents are held in Document Type Definition (DTD) or XML Schema Definition (XSD) documents.

An XML document can be described as 'well-formed' and also as 'valid'. A well-formed document is one that conforms to the physical structure of XML but it has not

been validated against a schema. A well-formed XML document does not need to conform to a schema. A valid XML document is a well-formed document that has been verified against a specific schema. Each node in the document conforms to the rules of the specified schema.

The extensibility of the XML language means that new elements or attributes can be easily added. If a client application does not recognise a field, it is simply ignored. An XML document can therefore contain much more information than a particular client application requires but this excess information does not render the client inoperative. However, it should be noted that if new elements or attributes are added to a document that previously conformed to a specified schema then that document will no longer conform unless the schema is also updated. Moreover, unknown items are retained within the document when the client application stores it. XML allows for the changing or addition of individual nodes in a document without the need to rewrite the whole document.

One advantage of the schema language is that the schema does not have to be rigorously defined before implementing a related XML document. Instead, an XML document can be developed and the schema automatically inferred from the structure of the document. If the document is subsequently modified, the schema can be updated automatically. This allows a more flexible approach to the development and enhancement of a document without losing the conformity checking tools.

3.3.4 Namespaces

XML allows the definition of data using tags appropriate to the document structure. It is possible that two different schemas might use the same tag name to describe completely different data definitions. In such a case this name collision would likely cause errors in parsing a document. Namespaces overcome this problem by associating a tag name with a target namespace, which uniquely identifies and differentiates tags from different schemas. The only requirement is that a target namespace be unique and this is typically a URL to a unique domain. In XML documents it is usual for the full namespace to be abbreviated to a short prefix. These prefixes are required to be unique in a specific instance of a document but do not need to be unique across time and space.

For example, Figure 3-2 shows a fragment of a FireBaseXML Conversion document (see Section 6.3.2 for further details) in which line 2 specifies two target namespaces; one for the XSL transformation schema and one for the FireBaseXML Conversion schema. Each target namespace is also given a unique short prefix, in this case xsl and hrrt respectively although they could be anything else if desired by the fragment author. These prefixes are then applied throughout the document to the element tags.

```
1: <?xml version="1.0" encoding="UTF-8"?>
2: <xsl:stylesheet version="1.0"
       xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
       xmlns:hrrt="http://www.civil.canterbury.ac.nz/spearpoint/HRR_Database/
       FireBaseXML_Conversion.xsd">
3:     <hrrt:Transformation>
4:         <hrrt:name>Raw</hrrt:name>
5:     </hrrt:Transformation>
6: </xsl:stylesheet>
```

Figure 3-2. XML document that uses namespaces.

3.4 Transformations

The XSL language controls how an XML document is output and can be used to transform from one format to another e.g. XML to HTML. XSL contains a pattern query language that can be used to address and filter the elements and text of an XML document. Portions of an XML document can be extracted using an XSL pattern and these patterns can contain items such as references to particular attributes, wildcards and regular expressions. XSL Patterns are being replaced with the XPath language though either can be used to query an XML document.

One of the aims of XML is to enable the efficient sharing of data and documents by agreeing on a single schema for a specified domain of interest. In order to promote interoperability between distinct XML vocabularies, transformations are required. The XSL Transformations (XSLT) language describes these transformations and can be used to transform an XML document into any other text-based document such as XML, HTML, CSV and so on. Figure 3-3 shows the relationship between an XML

document, its schema (where one exists) and how an XSLT can be used to create a new document.

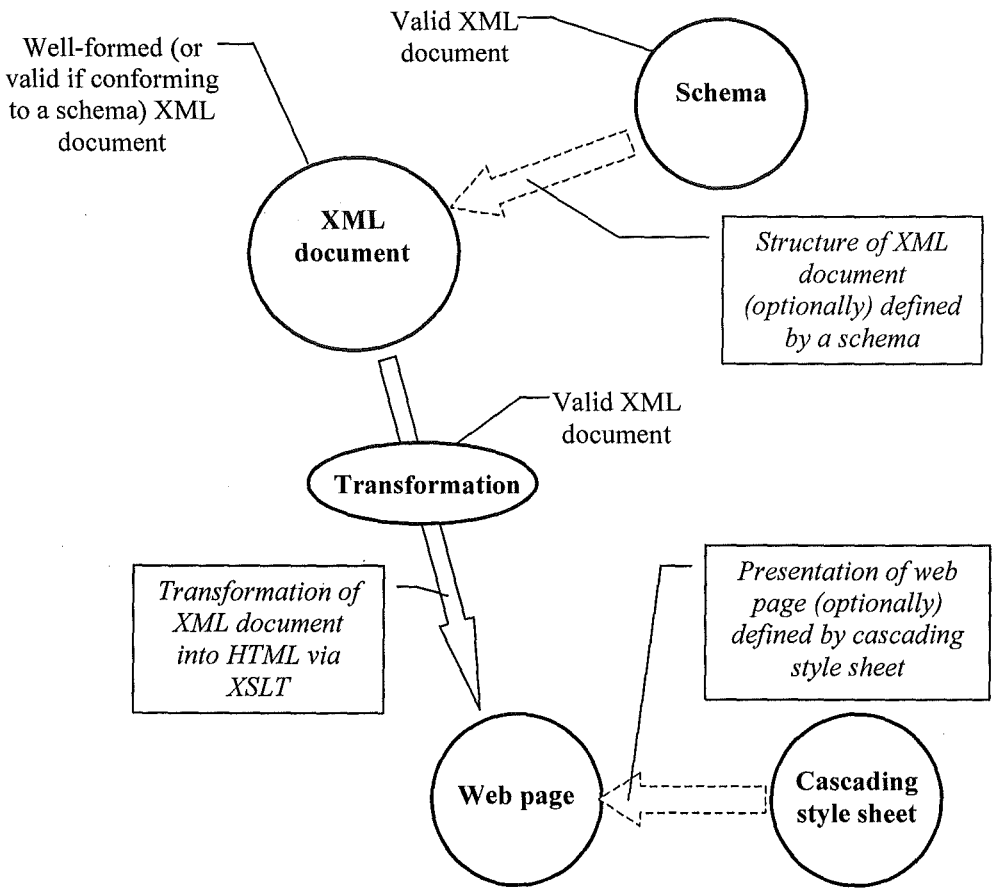


Figure 3-3. The relationship between an XML document, its schema and a transformation to another document (in this case a web-page).

Consider a situation in which there are two applications that want to share data. In many cases the author of the transmitting application may not have the source code for the receiving application and visa versa. In this situation, either the transmitting application would have to reformat its output data in a form that can be interpreted by the receiving application or the receiving application would have to interpret the output from the transmitting application. In either case, the structure of the required data may not be clear. For example, what do the data represent, what units are the data in, is the order of the data important etc? Furthermore, if either the transmitting or the

receiving application is modified then this may require modification to the data transfer process.

Using XML-based technologies, the authors of a receiving application could provide an XSL transformation to convert an XML document containing the required data into the format required by that receiving application. This approach has several advantages:

- New transformations can be made available to the transmitting application on a server for easy access;
- If the receiving application needs to have its input data structured in a new format then an altered transformation can be published by the authors of that client application.

An important aspect to emphasise is the independence of the XML document, the receiving application and the XSL transformations. The XML document is an independent entity that conforms to a specified schema. The client application can process XML documents that conform to the schema. One or more clients could use the XML document for completely different purposes. The XSL transformations are independent of the client application and these can be applied to a suitable XML document. The client application, the XML document, the schema, each XSL transformation and the transformation schema may all be on the same machine or on one or more remote machines connected over the Internet. This approach is used during the development of the rate of heat release database described in Chapter 5.

3.5 XML document inter-relationships

It is essential to realise that XSL style-sheet documents and XSD schema documents are also valid XML documents that conform to specified schemas as illustrated in Figure 3-4. This means that an XSL style-sheet can be applied to any of these documents. In other words a style-sheet can be applied to a style-sheet document even to the point where a style-sheet applies to itself. It is also important to realise that schemas are described as XML documents which themselves have their own schema (effectively a "schema of schemas"). Therefore XSL transformations used to process

and view XML documents can equally be applied to view and process XSD schema documents. This inter-relationship makes the XML language extremely powerful.

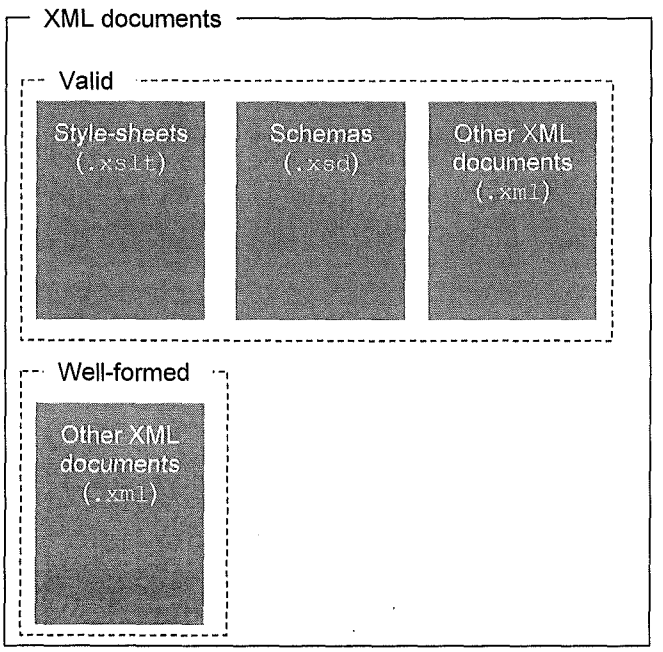


Figure 3-4. Relationship between Style-sheets, Schemas and XML documents.

With the development of database structures, it is important that the structure is properly documented and that documentation is readily available to any developer who wants to use the structure. In the past, the database structure and the documentation have been separate entities. For example, the documentation of the structure was held as a paper-based or electronic file that was physically separate from an actual database. Inevitably, during the development of the database structure, the documentation lagged behind the structure. Any user of the database structure had to be able to obtain this documentation, which meant knowing where to access it and knowing whether the documentation was up-to-date. Here lies one of the powers of this technology where XML, XSL and XSD documents are closely inter-related. Effectively such documents are "self-documenting" and allow the documentation for a database to be included in the database document. It is essential to note that only the syntactic structure of the data can be found from the structure of the XML document and some form of human entered and human readable natural language wording would still be required to describe the semantics of the database. The XML structure

does allow for this semantic description to be resident with the document itself thus keeping the database structure and semantic description to be held in a single place.

Chapter 4:
THE POTENTIAL IMPACT OF BUILDING PRODUCT MODELS ON FIRE PROTECTION ENGINEERING

Spearpoint M.J. The potential impact of building product models on fire protection engineering. Fire Protection Engineering, Issue 19, pp.42-48. 2003.

Paper Referees:
Journal Editorial Board.

Abstract

Information exchange can be an issue common to any area of modern life where computers are used to store and manipulate that information. The ability to efficiently exchange information increases productivity and reduces errors. Recently, and in particular with the explosion in the use of the Internet, this topic has emerged as an area of particular importance. In this article we try to answer a number of questions related to building product models and place them within the context of fire protection engineering.

4.1 What are the problems associated with current computer-based information exchange?

Without a standard format for the content and transfer of information between software tools, conversion processes are necessary. Each conversion process may 'devalue' the information as the content has to match the lowest common format. Furthermore, ambiguities may occur in the data that cannot be resolved during the conversion. The use of software tools across a whole range of engineering disciplines means that interoperability between these tools is becoming of critical concern. Building product models provide a means in which efficient information exchange can come about.

4.2 What is a building product model?

Fully automated, one time data entry and seamless integration of project life cycle work processes can be identified as a significant trend for the construction industry which have the potential to revolutionize the industry. The construction process covers the complete life-cycle of the structure, from inception to demolition. Fire protection engineering is only one domain of many that make up the overall construction process. Additional domains might include architecture, structural engineering, environmental engineering, building services and many others. Many parameters related to a structure are common to a range of disciplines. These parameters may include the building geometry and topology, the materials and components used in the construction and the location of the structure within the broad environment.

In general, any product can be considered to consist of a collection of 'entities'. A 'product model' expresses the type of entities that represent the product; the properties that are needed to describe those entities and the inter-relationship between entities. A 'building product model' is a product model that specifically relates to buildings where entities may be physical objects such as doors, windows, walls etc. or more conceptual entities such as spaces or processes. Within a building product model a door entity has specific properties (such as its dimensions and construction material/s) and the relationship with the wall in which it is located (its position, orientation etc.).

Conventional software tools are not able to describe the performance or properties of the entities and their component parts. A common feature in all mainstream CAD packages is the ability to transfer files in Data Exchange Format (DXF). However DXF files are only able to represent entities as a collection of points and arcs. They are not able to carry additional information or parameters such as density, thermal conductivity etc. in a standardized way. The new generation of CAD tools overcome these limitations through the use of object-oriented technologies. Entities are described by using properties, which could include geometric information that can be rendered graphically, and other information relevant to that entity.

There is considerable work at an international level, both within the construction industry and in commerce in general, that is developing methods for storing, transmitting and manipulating meaningful product data in an open environment. Without such methods, the construction industry will continue to be at a disadvantage because of the lack of integration between its various proprietary systems. Projects will continue to require labor-intensive and error-prone manual interpretation with the re-entry of information at the interfaces between different partners and across the boundaries of work processes.

4.3 What is the IFC Model?

Many of issues discussed above are already being addressed through the International Alliance for Interoperability (IAI), a world-wide group of engineering professionals, software developers and researchers who are developing a building product model,

referred to as the Industry Foundation Class (or IFC) Model, that permits an object-oriented description of many aspects of buildings and related services (Figure 4-1, adapted from Liebich and Wix, 2000).

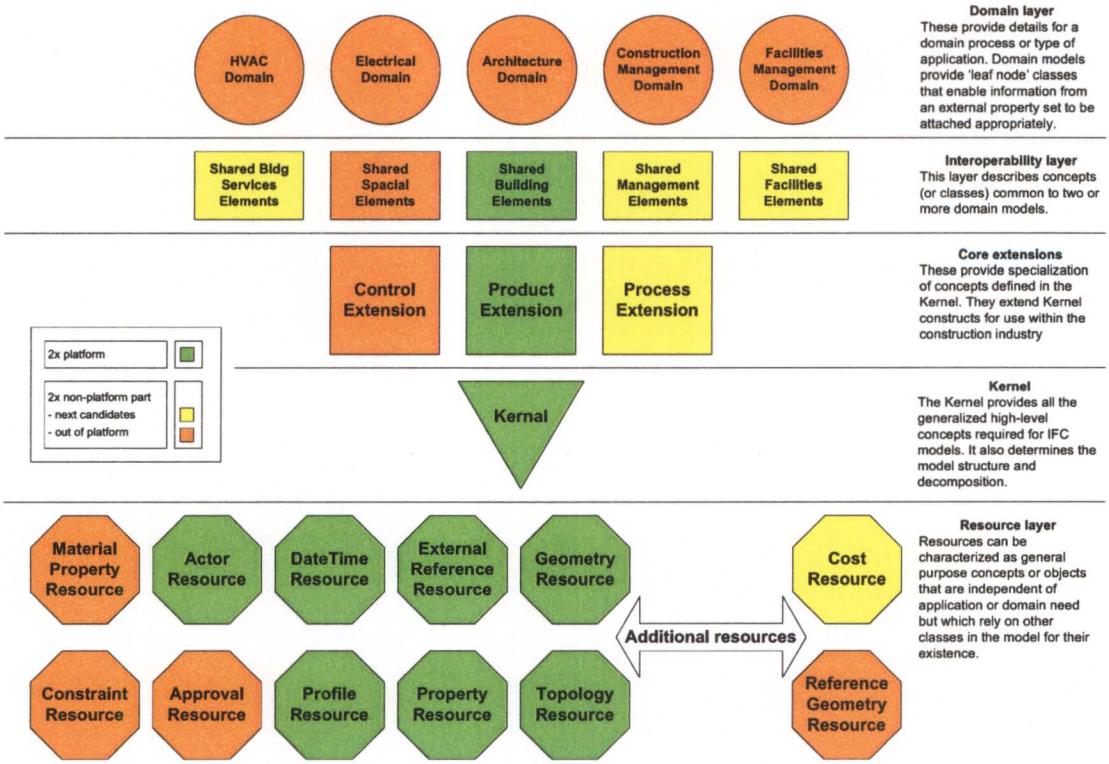


Figure 4-1. The IFC Model 2x architecture.

The IFC Model development began around 1996 and is an extension of a number of earlier projects. It is not intended that the IFC Model should describe every aspect of a building down to the detail required by all of the engineering disciplines involved in the construction process. This would make the building product model too unwieldy. Instead, it is designed to describe the most common parameters that may be required.

In order to facilitate the inclusion of additional properties, the IFC Model includes a mechanism referred to as 'property set definitions'. This mechanism allows the IFC Model to expand on the properties that characterize an entity beyond what is included in the IFC Model. A property set definition allows for the sharing of standard sets of

values across entities and for the definition of different property values within individual copies of an entity.

4.4 What does this all mean for fire protection engineering?

There are several potential ways in which the development of building product models might enhance the work of fire protection engineers and some of these are discussed here. Although some of these concepts are not necessarily new to fire protection engineering (Mowrer and Williamson, 1988) the latest developments in the technology allow these ideas to be implemented now.

4.4.1 Improved exchange of building geometry

Building product models will facilitate the automated import of building plans into computer calculation tools. Although some currently available computer calculation tools can read CAD floor plans, their facilities are limited. For example, the **SIMULEX** egress model (Thompson and Marchant, 1995) has the ability to read DXF files but these files generally need to be edited manually prior to using **SIMULEX** because of the limitations of the DXF format. In the future, building descriptions will be exchanged in such a way that computer models can more effectively make use of the data.

4.4.2 Property set definitions

Manufacturers of fire protection hardware could publish property set definitions on their web servers using industry agreed classification for the specific products. The property set definitions would contain essential information required to characterize the product such as physical dimensions, performance metrics and listing documentation. The definitions may also include optional characteristics and characteristics unique to a particular manufacturer. Thus for a sprinkler we might want to provide details such as the model number, dimensions, Response Time Index (RTI), temperature rating, construction material, organizations that have listed the head etc.

4.4.3 Standards, codes and certification

Instead of being static paper documents, standards and codes could soon be published as dynamic electronic documents. This form of publication could lead to the automatic incorporation of relevant code requirements in the design process and documentation. Furthermore, the electronic publishing of listings and certification will allow up-to-date verification that a product meets any specific regulatory requirements.

4.4.4 Fire test databases

Fire test results and fire-related material properties can be published electronically. These data can be imported into an electronic building model using property set definitions.

4.5 How might a building product model and property set definitions be used?

Let us imagine that a fire protection engineer wants to assess a sprinkler system using a computer fire model in order to examine specific fire scenarios using a building product model and property set definitions. An architect has already created a description of the spaces in a document that uses a specified building product model. The fire protection engineer can use this document in order to complete their assessment in association with other relevant parties and then pass the revised document on through the design process. Thus the building product document grows as the design proceeds, with new information being added as tasks are carried out.

In creating the fire scenarios, the fire protection engineer might need the rate of heat release from the furniture items that have been identified by the interior designer and specified in the building product document. At this point the fire protection engineer could access a database of fire properties in order to select an appropriate design fire for the furniture. The heat release data is extracted from the database and appended to the furniture entities in the building product document as a property set definition. The computer fire model now obtains the building geometry, furniture properties (which includes the rate of heat release) from the building product document. At this stage, the fire protection engineer might also obtain additional properties from an HVAC engineer (such as air movement due to the ventilation system), the properties

of the sprinklers they intend to use and information from the AHJ relating to any code requirements. Since the sprinkler manufacturers publish the property set definitions of their sprinkler hardware in a format that is compatible with the building product model, the fire protection engineer can directly import the relevant performance metrics such as the temperature rating and RTI into the computer fire model.

Once the fire protection engineer has completed the modeling and decided on an appropriate sprinkler head layout, the building product document can be passed to the sprinkler installer. The sprinkler installer could then use a hydraulic design tool to determine pipe schedules, again using the building product document to obtain pertinent information supplied by the fire protection engineer and the sprinkler manufacturer. The completed sprinkler network is added to the building product document ready for the quantity surveyor to generate bills of quantities. Again this is done through the building product document by efficiently identifying the required pipe lengths etc. Finally the bill of quantities can be related back to the sprinkler manufacturer in order that a contractor can deliver the correct hardware onto the construction site. Figure 4-2 illustrates the above sprinkler design and delivery process showing the linkages between each step.

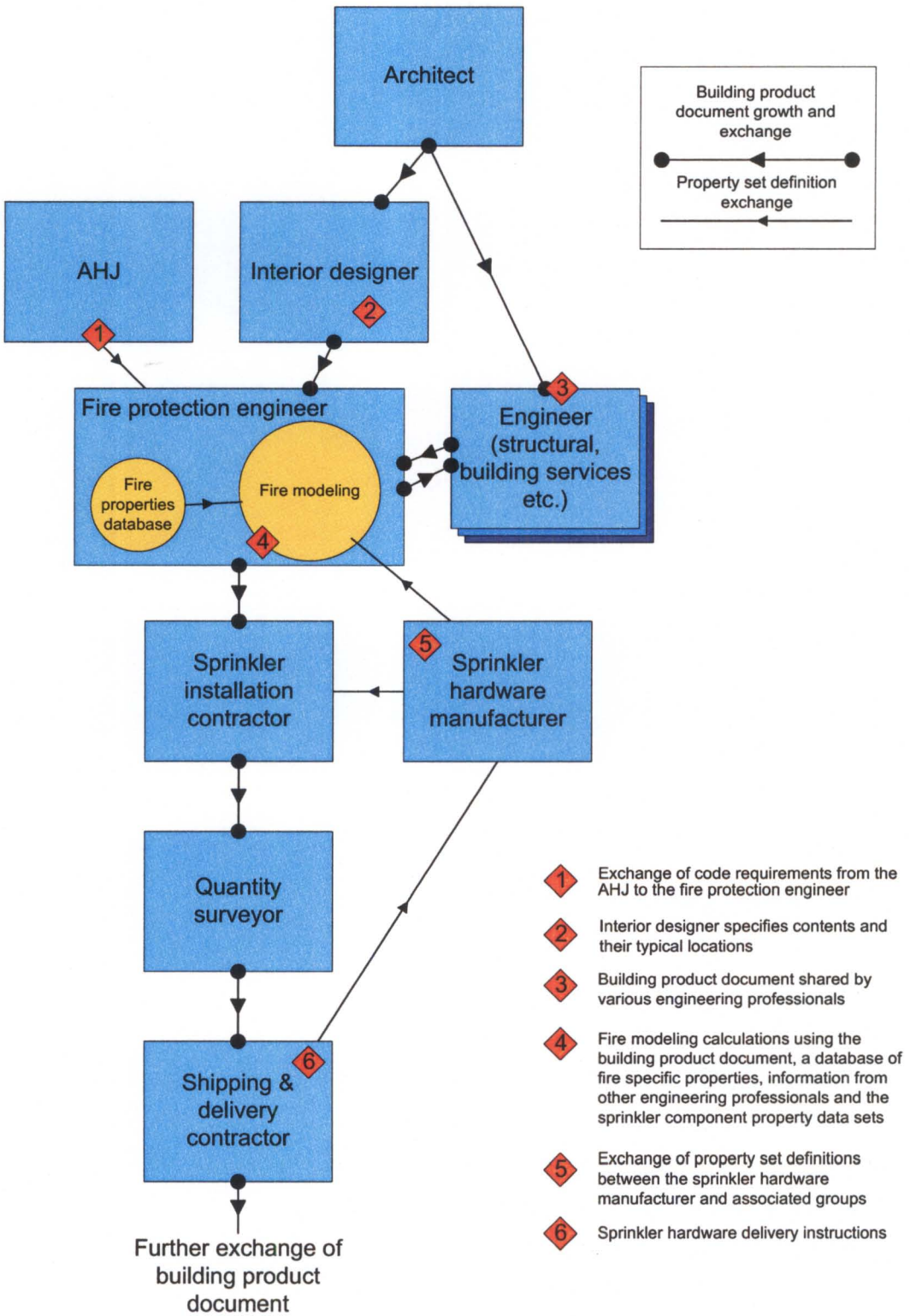


Figure 4-2. The exchange process for a building product document and property set definitions.

The above description is only one way in which a building product model could be used. The example describes a linear process whereas some tasks might take place concurrently or at different stages of the design process. The important aspect is that a single document is used throughout where each participant in the process uses information supplied by others and adds their task specific data back into the building product document.

4.6 Where are we now with building product models?

The above hypothetical sprinkler design scenario is still somewhere in the future. Some of the tools that are required to realize the above scenario are already available or under development whilst others are a considerable way off.

The IFC Model – The current version of the IFC Model (2x) includes details of the geometry and topology of a building and identifies walls, windows, doors, furniture and HVAC entities, many of which are useful to fire protection engineers. The model has a very limited set of properties relating to fire protection engineering. For example, walls, doors, and windows can be assigned a fire resistance rating; fire and smoke dampers are included in the model; stairs can also be given a fire resistance rating and declared as exit paths; and insulation materials have a flammability rating property.

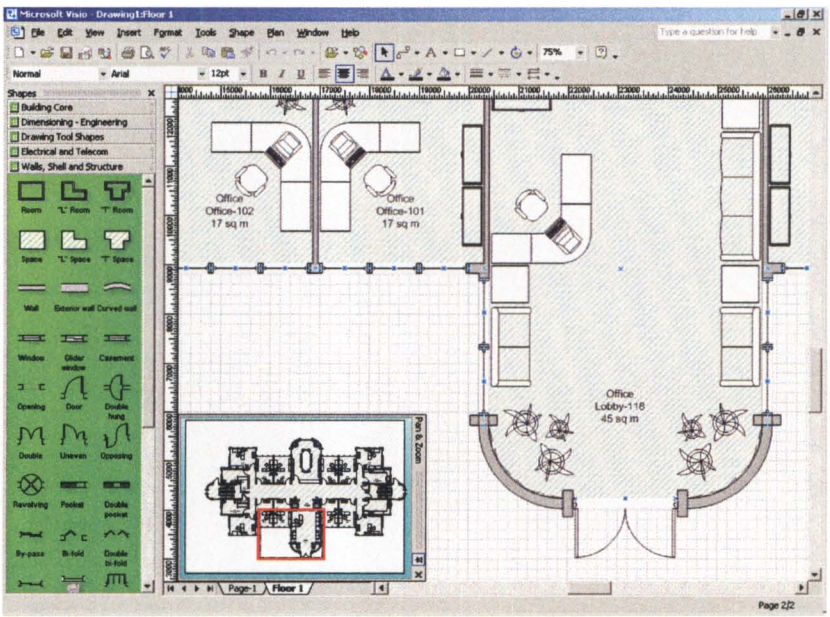


Figure 4-3. CAD software tool – 2D plans of building and contents (Document simple2_001204.xml available from the BLIS project website at http://www.blis-project.org/).

IFC compliant tools – There is an ever-increasing range of software tools appearing that are able to exchange IFC documents. These tools currently include CAD (Figure 4-3), thermal design, quantity take-off, model consistency checker (Figure 4-4, Graphisoft, 2001) and others. There are also a significant number of tools in development or under test including HVAC design, energy simulation & code checking, electrical system design and the list goes on. In terms of fire protection engineering, very little has been done so far. Preliminary work has already been undertaken at the University of Canterbury into a tool to interface IFC Model documents to **CFAST**.

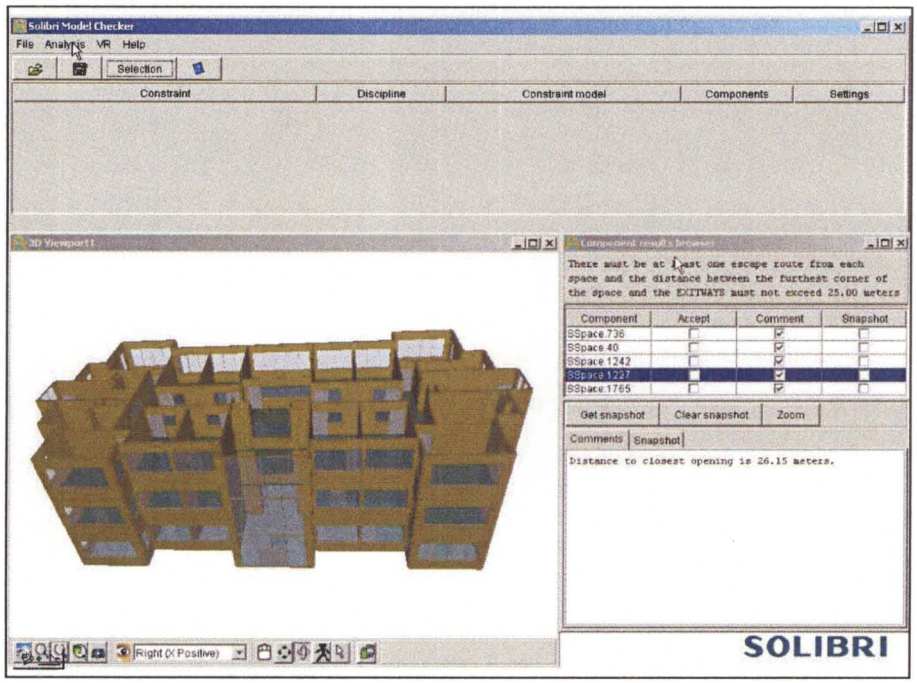


Figure 4-4. Design model checker - Checking construction rules in a building model.

Codes & standards – Currently in Australia there is a move to provide the next edition of the Building Code of Australia (BCA) in an electronic form. This will allow automatic searching of the code for particular clauses relevant to a discipline or specific building component. It will mean that the BCA could be viewed on-line such that all clauses relevant to a particular discipline or sub-discipline could be easily extracted.

Property set definitions – so far there is be little work on developing property set definitions for fire protection engineering related components. Some work has begun on providing rate of heat release information suitable for incorporation into the IFC Model (Chapter 5) and it is hoped new initiatives will expand on this work.

4.7 What does the future hold?

There is still much work to do before seamless electronic data exchange becomes widely available. The IFC Model contains only a certain level of detail regarding many domains and there is only a limited amount of information that relates to fire protection engineering. However, the IFC Model already has a rich description of the fundamentals of buildings and the development of domain-related information is

proceeding through international efforts. Software tools that are IFC compliant are now available and many others are under various stages of development. As more tools become available, so will the demand that additional tools be able to exchange IFC files increase.

The next release of the IFC Model will deal with Facilities Management, Structural Engineering, Codes & Standards and Building Services. There is some work already being undertaken to describe specific building products using property set definitions. Other issues such as the contractual and legal aspects of using electronic building models; the ability to concurrently share data; and the development of a lexicon of building terminology are also being investigated.

At this stage, it is important that the fire protection engineering community be aware of the developments in building product models so that we do not get left behind. Developers of fire-related software tools need to assess whether they should be enhancing their programs to read and write building product models such as the IFC Model documents. Manufacturers of fire protection related hardware might consider the formulation of agreed property set definitions. Regulators might want to consider alternative means of publishing codes and standards. The use of Information Technology and computer-based software tools will continue to grow in both the construction industry and more widely. Building product models and their associated technologies will play an important part in integrating this growth.

4.8 Additional comments

The following paragraphs were removed from the original paper in order to make the document fit the publication length limitations or subsequently added to provide further details. This text gives useful background and so has been included here for completeness albeit in a slightly modified form from the original submission.

4.9 What other related work is there?

The IFC work uses STEP (Standard for the Exchange of Product model data) technology, ISO 10303. STEP consists of five main categories of standards (Nell, 2001):

1. Description methods – the definitions universal to STEP
2. Implementation methods – the mappings from the formal specifications to implementation representations
3. Conformance testing methods – methods to test the conformance of software tools to STEP
4. Common resources – generic resources that can be used by application protocols
5. Application protocols – data models used to describe a specific product application.

The web is changing the way in which information is stored and exchanged. Aspects such as document sharing, online product information, data repositories and mark-up languages have all seen rapid developments with the expansion of the web.

One rapidly emerging technology that has appeared recently is the Extensible Mark-up Language (XML) language. XML is a meta mark-up language that provides a format for describing structured data. XML is similar in many ways to HTML and HTML can be viewed as a special subset of XML. However, unlike HTML, XML separates the data from its presentation and processing which allows the integration of data from diverse sources.

There is a close relationship between XML, the STEP language and the IFC Model. ISO 10303 Part 28 is an XML implementation method and the IAI have ifcXML which specifies the translation process of the IFC Model into XML format. Other areas in which XML is being used include the representation of information in the Architecture, Engineering and Construction (AEC) industry and a product model that facilitates the exchange of data created during the land planning, civil engineering and land survey process. XML is also being seen as the ideal way in which to create property set definitions.

The development of IT in the construction industry is a rapidly changing environment. It is recognised that specifications for data description and exchange will evolve over time. The current industry-wide approach appears to be that it is preferable to evolve this technology in stages rather than to conduct an extensive study that may require many years before anything becomes available for use by the industry.

4.10 How can property set definition data be accessed?

Property set definition information would need to be accessed through the use of electronic and online systems to populate the data requirements of building product models. Several independent initiatives by commercial organisations and standardisation bodies have been undertaken to enable the integration of manufactured product data with design tools. However, these electronic catalogues suffer from the limitations found in paper-based catalogues which then requires human intervention. A number of barriers to the ongoing development of electronic product catalogues have been identified by Amor, Jain and Augenbroe (2004). These barriers include agreement on the parameters associated with a vast number of available products, extraction of those parameters from manufacturer's information resources and the ability to select appropriate product data from one or more catalogues.

The IFC property set definition mechanism allows new properties to be allocated to an IFC entity. However the mechanism has disadvantages in that there is no formal specification for a product outside what is published in the IFC Model. Thus differing specifications for the same product could be introduced by independent bodies that do

not work together with a common goal. Currently, at least in terms of fire engineering related products, there are not commercial sources of electronic information that follows the IFC property set definition specification.

Chapter 5:
THE DEVELOPMENT OF A WEB-BASED DATABASE OF RATE OF HEAT RELEASE MEASUREMENTS USING A MARK-UP LANGUAGE

Spearpoint M.J. The development of a web-based database of rate of heat release measurements using a mark-up language. Proc. 5th Asia-Oceania Symposium on Fire & Technology, Newcastle, Australia. 2001.

Paper Referees:

Three anonymous referees.

Abstract

The application of most computer-based fire models is dependent on the user supplying the rate of heat release data that describes the design fire for a chosen scenario. Having access to a database of rate of heat release measurements assists the user in making their selection. The ability to use that data in a wide variety of general and specialised computer tools enables effective use of the data. This paper examines the current developments in database implementation using mark-up languages and how they can be applied to engineered fire protection design. In particular, the increased use of XML technology as a method of information storage, retrieval and manipulation in several engineering related fields are discussed since these developments are very likely to have an impact on fire engineering.

An XML-based schema for the implementation of a database of single and multiple item rate of heat release measurements is presented and an online database has been created using the schema. The database and its underlying schema can be viewed in a web browser. The database can also be queried via a client program using broad search criteria; the relevant matches can be viewed graphically and a selected dataset can be extracted in a format suitable for further processing. A number of transformations have been developed that allow a selected dataset to be converted from XML to an alternate format suitable for commonly used fire models or more general computer software tools.

Although the database currently operates as a stand-alone entity, the work is also aimed towards integration with the developments taking place in interoperability between a wide range of engineering-related software tools. This paper shows where this web-based database fits in with these developments.

5.1 Introduction

5.1.1 Data selection

When using most of the currently available computer-based fire models, it is the effect of the rate of heat release that has the greatest impact on the calculation of the conditions within a compartment (Babrauskas and Peacock, 1992) and therefore the selection of appropriate data is of primary importance.

In many fire-modelling situations the scenario that is being examined does not include a burning item (or items) that exactly matches those that have been tested and measured in the laboratory. Therefore, when searching for an appropriate heat release curve for use in a fire model, the user may not have a specific item or test in mind but may be looking for an item that best matches their particular scenario. For example, in the study by this author (Spearpoint et al., 1999) the scenario to be modelled included a loveseat but it did not exactly match any of those loveseats that have been tested and reported in the literature. In order to model the scenario, a nearest match item was identified and the heat release rate for that item was used as input to the fire models.

5.1.2 Data exchange

The exchange of data between various electronic tools can be a problem common to any area of modern life where computers are used to store and manipulate data. Recently, and in particular with the explosion in web-based information exchange, this area of data exchange has been of particular importance. Without a standard format for the content and transfer of data between software tools conversion processes are necessary. Each conversion process may 'devalue' the data as the content of the data has to match the lowest common format. Furthermore, ambiguities may occur in the data which cannot be resolved during the conversion. The use of software tools across a whole range of engineering disciplines means interoperability between these tools is becoming a critical issue. The ability to efficiently exchange data increases productivity and reduces errors.

5.1.3 Rate of heat release catalogues

Although there is a range of rate of heat release data available in the published literature it can be time consuming to find the data for a desired item and it may require the user to digitise the heat release curve before it can be used in a model. The concept of having a database of test data and also software to convert that data into different formats are not new ideas. Several attempts have been made at achieving both. Examples of databases include those found accompanying fire models as part of the package (Bukowski, 1989; Deal, 1995), the work of Särdqvist (1993), the **Fire Data Management System** (**FDMS**) (Portier et al. 1997) and the online database initiated at NIST (2001). An example of conversion is the **DCS** program developed at SINTEF (Lønvik and Opstad, 1992). Previous unpublished work by this author (Spearpoint, 1993a) of a database of heat release curves and a conversion program also fit into these examples. It is not necessarily the intention of the work described in this paper to replace this earlier work, but instead to examine how the latest developments in information technology can be applied to the storage and exchange of such material.

5.1.4 Requirements

A number of requirements for a database of rate of heat release measurements can be identified:

- The use of a standardised data description technology for the database structure.
- Make the database widely available and easily accessible.
- Allow broad search criteria to be applied to the database.
- The ability to view, extract and process selected data using a variety of general and specialised software tools.
- Allow the scope of the database to be extended without making earlier software redundant.
- The ability to integrate the database with other tools across a range of disciplines.

The success of previous efforts to create catalogues of rate of heat release measurements and tools to manipulate them were limited by the technology available at the time. The web provides a convenient resource for storing and distributing a

database of rate of heat release measurements and the development of associated mark-up languages offer potential ways in which the above requirements can be met.

5.2 Extensible Mark-Up Language

5.2.1 Specification

Extensible Mark-up Language (XML) (Light, 1997) is a meta mark-up language that provides a format for describing structured data. This facilitates more precise declarations of content and more meaningful search results. XML is defined by the World Wide Web Consortium (W3C) and is independent of software tools or vendors. XML is similar in many ways to HTML and HTML can be viewed as a special subset of XML. However, unlike HTML, XML separates the data from its presentation and processing which allows the integration of data from diverse sources.

There are a number of advantages in using XML as the basis for a database:

- The extensibility of the XML language means that new fields can be easily added. If a client application does not recognise a field, it is simply ignored. An XML document can therefore contain much more information than a particular client application requires but this excess information does not render the client inoperative.
- The XML document can be easily and efficiently searched via a pattern query using a number of techniques and programming languages. Libraries and parsers for creating and manipulating XML are already available.
- The document structure is in a human readable form allowing it to be edited in a text editor. However, more sophisticated editing tools specifically designed for XML are also available.

5.2.2 Document structure

An XML document consists of a collection of nodes. Nodes can be one of several types including elements or attributes. Unless an element is empty, it consists of a pair of tags plus its content. XML allows the definition of an unlimited set of tags and XML provides a data standard that can encode the content, semantics and schema of a dataset. Elements can have any number of sub-elements nested to any depth and any amount of text can be included. Additional information can be added to an element

by using attributes. Attributes have a name and a value. Values are optional and an attribute can exist as a name only.

5.2.3 Schema

The schema describes the structure of the XML document in terms of the relationship between elements and attributes and the types of data that can be stored by them. A specific XML document can be automatically checked against a schema for conformity. An XML document can be described as 'well-formed' and also 'valid'. A well-formed document is one that conforms to the physical structure of XML but it has not been validated against a schema. A valid XML document is a well-formed document that has been verified against a specific schema. A schema can be automatically inferred from the structure of an XML document. If the document is subsequently modified, the schema can be updated appropriately. This allows a more flexible approach to the development and enhancement of a database without losing the ability to perform conformity checks.

XML Schema is an agreed "schema of schemas" published by W3C (World Wide Web Consortium, 2001). It is an XML document that defines the content and semantics of particular vocabularies and the structure of XML documents that use those vocabularies.

5.2.4 Transformations

One of the primary aims of XML is to enable the efficient sharing of data and documents by agreeing on a single schema for a particular field of interest. However, the likelihood of a single vocabulary fitting the needs of all interested organisations is very small. In order to promote interoperability between distinct XML vocabularies, transformations are required. Extensible Style-sheet Language Transformations (XSLT) describe these transformations and can be used to convert an XML document into another XML vocabulary or any other text-based document such as HTML, comma-separated values (CSV) etc.

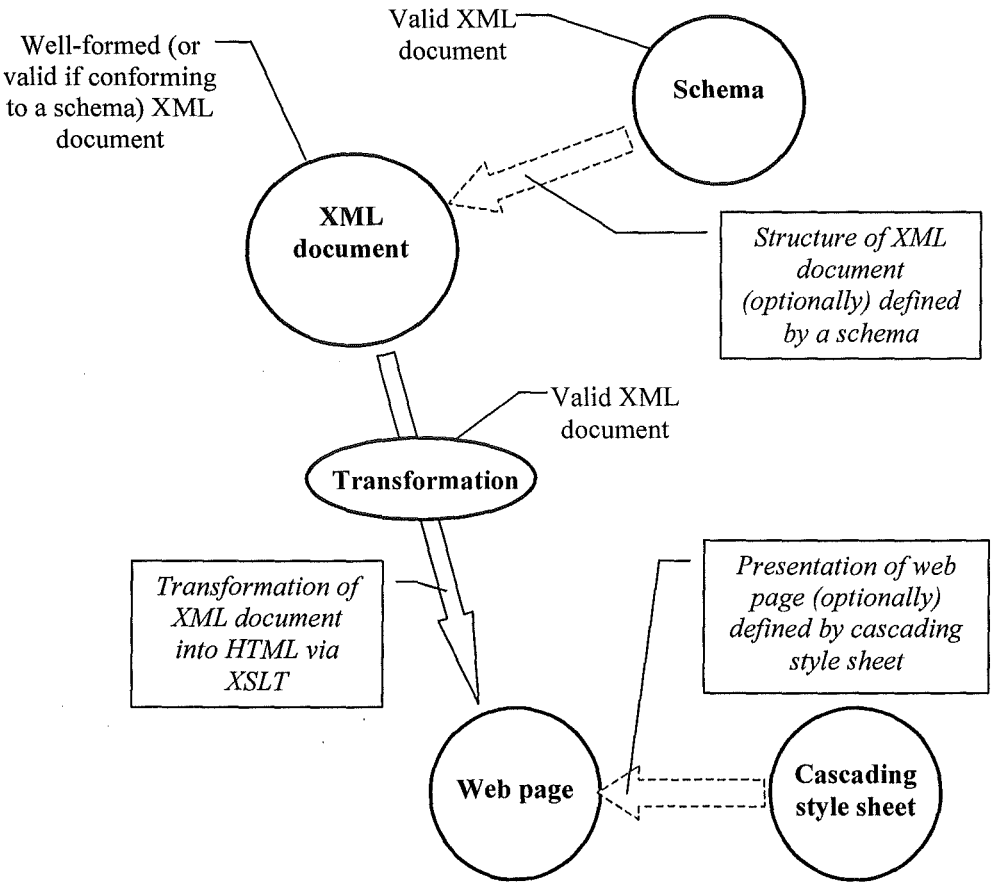


Figure 5-1. The relationship between an XML document, its schema and a transformation to another document (in this case a web-page).

Figure 5-1 shows the relationship between an XML document, its schema (where one exists) and how an XSLT can be used to create a new document. Because of the way in which XML separates data from its processing and presentation, each component can exist remotely as separate entities. This ability to separate the components is one reason why XML technology is particularly suited to web-based data exchange.

5.2.5 XML use in engineering

The technical merits of XML discussed above have been recognised by several other engineering fields and developments in these areas will most likely have an impact on fire engineering. Most importantly is the work being carried out by the International Alliance for Interoperability (IAI) (Liebich and Wix, 2000). This international group

of engineering professionals, software developers and researchers is developing a family of schemas (referred to as the Industry Foundation Classes or IFCs) that permit an object-oriented description of many aspects of buildings and related services. Although their work does not use XML directly, they recognise its importance and implement an XML version of their schemas (Liebich, 2001). Other areas in which XML schemas are being developed include the representation of information in the Architecture, Engineering and Construction (AEC) industry (aecXML Working Group, 2001) and a schema that facilitates the exchange of data created during the land planning, civil engineering and land survey process (LandXML, 2001). These examples illustrate the diverse uses of XML technology in engineering and were an important consideration in the selection of XML for the development of the rate of heat release database described in this paper.

5.3 Rate of Heat Release Database Schema

5.3.1 Description

The development of schemas is one of the primary tasks required for interoperability. A schema for a database of rate of heat release measurements has been created and is referred to as the 'FireBaseXML' schema. It includes fields that describe a particular database plus one or more records (Figure 5-2).

Each record in the FireBaseXML schema can be broken down into three general components:

- The test data. This includes the rate of heat release data plus optional data for the initial mass of the item and its average heat of combustion.

- The test description. A short (one line) description of the test arrangement and a more detailed multi-line description of the test are stored with each record. Each record also has an associated set of 'item attributes' that describe the entry in generic terms. By using these generic attributes associated with each record, the user can make context meaningful searches for a particular item. For example a search can be made for all types of single seat chair, or alternatively for only upholstered single seat chairs.

The classification of groups and categories used in this database is similar to the BSAB format quoted by Särdqvist (1993). Items are classified into generic groups and then each group is sub-divided into a number of different categories. For example, groups include: 'chairs', 'easy chairs', 'love seats', 'sofas' and the chairs group can be categorised as 'metal', 'plastic' or 'wood'. This system of classification and the use of XML mean that additional groups and categories can be easily added.

- The source publication. Each record in the database includes details of the source publication from where the data was taken from. Where an online version of the source document is available, a link to the electronic document can be specified.

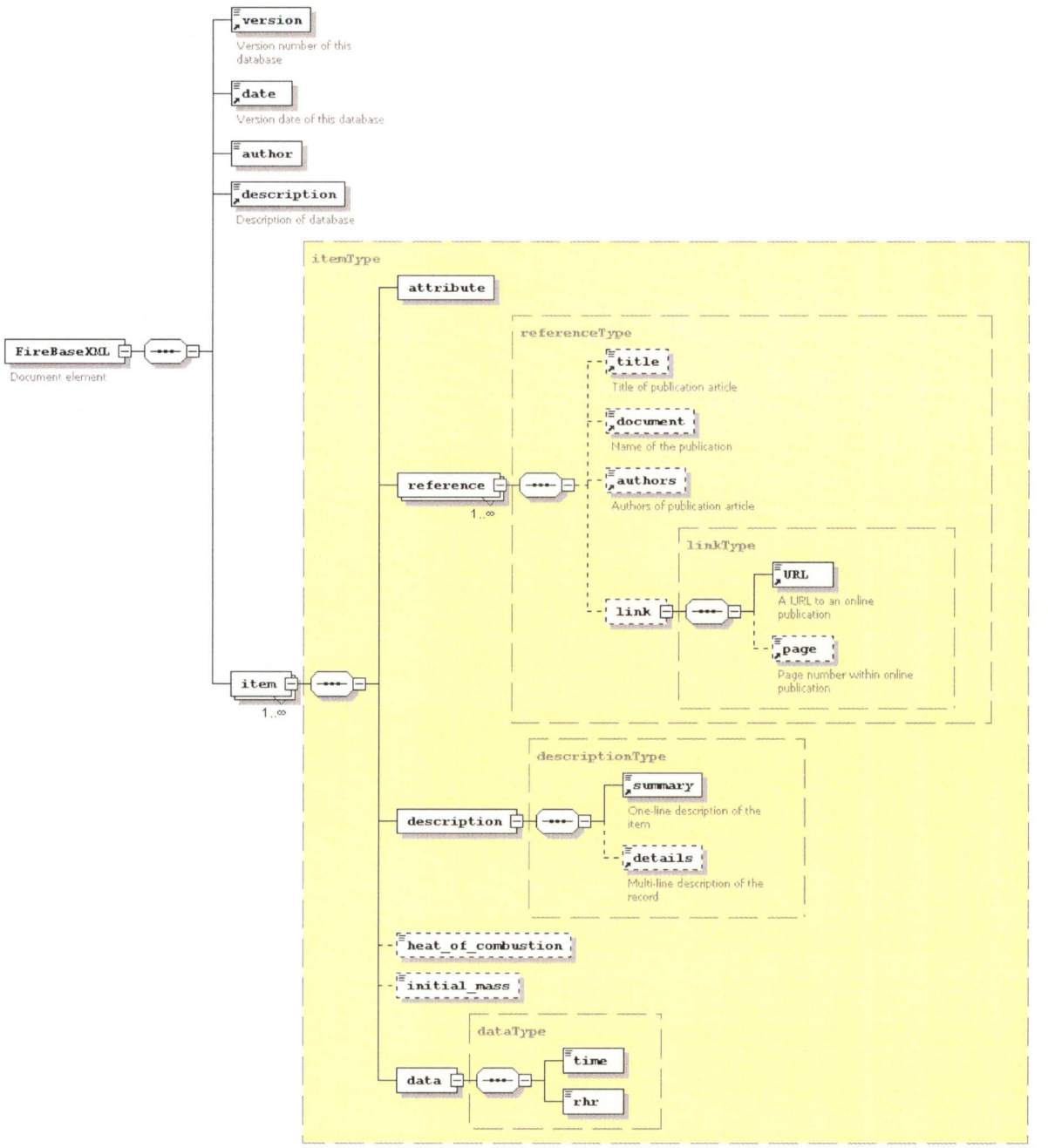


Figure 5-2. An overview of the FireBaseXML schema. Optional elements are shown by boxes drawn with dotted lines.

The FireBaseXML schema can be easily extended to allow for data obtained from small-scale test methods such as the cone calorimeter or for the inclusion of other suitable information such as toxic gas yields by defining additional nodes in the FireBaseXML schema.

5.3.2 Schema design

The flexibility of XML means that there is not necessarily only one way to develop a document structure so as to achieve a particular aim. In developing the FireBaseXML schema, assessments were made as to the most appropriate structure.

Specific attention was paid to the method in which the test data should be stored. Several different options were considered in which the level of detail provided by elements was examined (see Section 5.10 for a detailed discussion). A comma-separated format that provides sufficient descriptive detail but minimises the overall size of the document used. The general form of the structure for data is shown in a representational form in Figure 5-3.

```
<dataset>
    <entity name='T1'>
        <data type='def-type' units='def-unit'> 0, 1, 2, 3 </data>
        <data type='def-type' units='def-unit'> 20, 22, 28, 30 </data>
    </entity>
    <entity name='T2'>
        <data type='def-type' units='def-unit'> 0, 1, 2, 3 </data>
        <data type='def-type' units='def-unit'> 293, 298, 299, 305 </data>
    </entity>
</dataset>
```

Figure 5-3. Representational structure for data elements.

The 'def-type' attribute could be 'time', 'temperature' etc. and the 'def-unit' attributes associated with these types could be for example 's' for time and 'C' or 'K' or some other appropriately defined unit for temperature. The structure allows additional data types to be added to the schema by defining additional 'type' attributes. Units have to be explicitly defined so that ambiguity cannot occur.

The FireBaseXML schema was also designed to provide tight control over the specification of units to data but allowing for multiple forms of units if required. The units and measurement type are set as attributes on the main element. In this way the schema can be defined such that only valid measurement types and units can be associated to a particular variable. Figure 5-4 shows the schema fragment for the <heat_of_combustion> element.

Line 3 specifies the restriction for the content of the element as a number. Two attributes are associated with the element: "units" and "type" (lines 4 and 12). The units must be specified since their use is required and a list of restricted units are given in lines 7 and 8.

```
1: <xsd:complexType name="heat_of_combustionType">
2:     <xsd:simpleContent>
3:         <xsd:restriction base="xsd:number">
4:             <xsd:attribute name="units" use="required">
5:                 <xsd:simpleType>
6:                     <xsd:restriction base="xsd:string">
7:                         <xsd:enumeration value="J/kg"/>
8:                         <xsd:enumeration value="kJ/kg"/>
9:                     </xsd:restriction>
10:                </xsd:simpleType>
11:            </xsd:attribute>
12:            <xsd:attribute type="xsd:string" name="type"
                       use="fixed" value="available_energy"/>
13:        </xsd:restriction>
14:    </xsd:simpleContent>
15: </xsd:complexType>
```

Figure 5-4. The FireBaseXML <heat_of_combustion> schema fragment.

Additional unit enumerations could be appended to this list if necessary or the list could be reduced to a single enumeration so that only one form of unit is allowed. The type attribute (line 12) specifies the type of measurement and fixes a value to this type (in this case 'available energy'). Thus a valid <heat_of_combustion> that conforms to the schema would be.

```
<heat_of_combustion type="available_energy"
                    units="J/kg">15000</heat_of_combustion>
```

This schema structure means that association of an incorrect unit is automatically discovered during the validation of an instance of the FireBaseXML schema. Interrogation or extraction of the data can include a check of the units that the data is stored in and appropriate action taken to convert those units to another type if necessary.

5.4 Implementation

5.4.1 A FireBaseXML database

The rate of heat release database developed in this study is a valid XML document that conforms to the FireBaseXML schema. The database consists of a collection of rate of heat release rate measurements taken from various sources in the literature. Currently this database resides on a server at the University of Canterbury. However, one of the advantages of this web-enabled technology is that several organisations may want to publish their own databases using the FireBaseXML schema as their basis. Software agents could search one or more of these databases concurrently and return a combined set of results with any replicate information removed.

5.4.2 Web integration

The close association of XML and the web means that a FireBaseXML database can be integrated with an XML-compliant web browser. XSLT can be used to manipulate the database to dynamically generate web pages that allow a number of different 'views'. For example, Figure 5-5 shows a web page that lists the items in a FireBaseXML database and an item extracted from the database is shown as a stand-alone XML document.

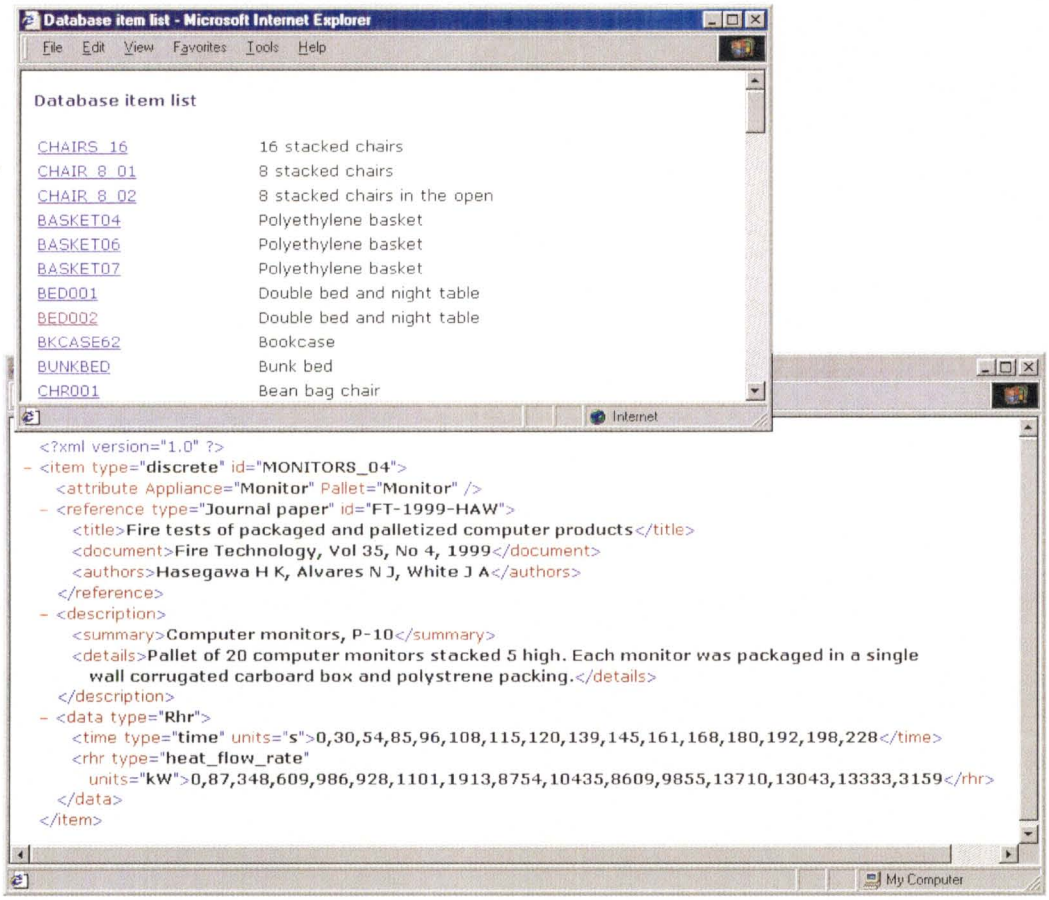


Figure 5-5. Web browser views of the records in a FireBaseXML database (top) and an extracted record (bottom) as an XML document.

5.4.3 Specific transformations

A number of specific XSLT documents have been developed so that records selected from a FireBaseXML database can be exchanged with general and specialised computer tools. These transformations are published on a web-server separate from any FireBaseXML database and remote from the eventual end-user. This has the advantage that a transformation can be delivered online at the time it is required. This means that the end-user does not have to ensure that their transformations are up-to-date. The transformation author need only upload a revised version of the transformation onto the appropriate server and this is delivered to the end-user when a transformation is requested. Currently transformations to process FireBaseXML records into files that can be used by a text editor, spreadsheet, **FPETool** (Deal, 1995) and a web browser have been completed. Additional transformations can be made available as the need arises.

5.4.4 The **SelectFire** client

The **SelectFire** program is a client application that allows a user to select a FireBaseXML database (either online or stored on a local disk) and then perform operations on that database. As a new database is accessed, the **SelectFire** program automatically scans it for the generic item attribute definitions allowing the program to dynamically construct the list of searchable item attributes. The user can therefore search for a particular generic group of items, view the rate of heat release curves of each match, consult any documentation related to a particular match if an online resource exits and make an appropriate selection (Figure 5-6). A selected match can be extracted from the database using a specified transformation so that it can be used in some further processing program whether that is a fire model or a more general package such as a spreadsheet.

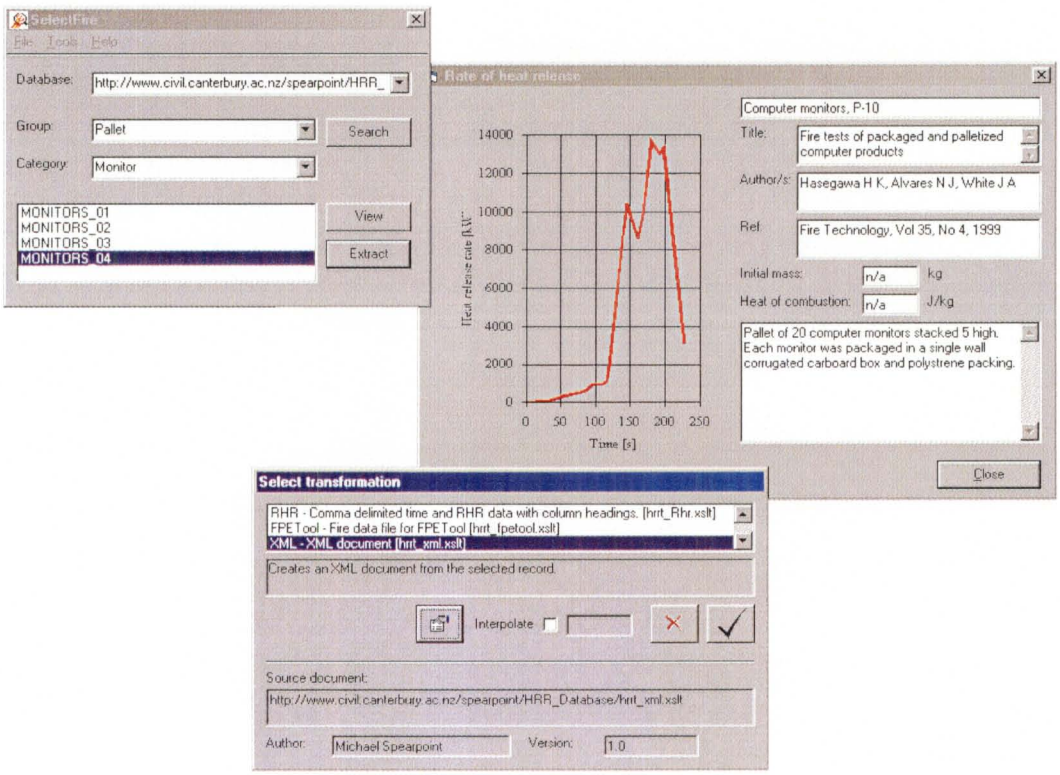


Figure 5-6. The **SelectFire** client with a selected record being viewed prior to its transformation.

5.5 Integration framework

Mowrer and Williamson (1988) proposed the integration of Computer Aided Design (CAD) information with fire specific properties of a building and its contents. The key features identified by Mowrer and Williamson required by CAD systems to permit integration included object-orientation, the association of attributes with objects and the ability to extract attributes from a CAD-developed drawing database.

The work described in this paper and that being carried out by the IAI mean that the opportunity to implement the proposals made by Mowrer and Williamson can be realised. A building and its contents can be described using the IFC schema and delivered as an XML document. The contents of the building can then be linked to an XML database of fire specific information. This can then all be delivered to a fire modelling application ready for any additional user input prior to computation. Although the development and implementation of the FireBaseXML schema is currently a stand-alone entity, it is intended that it will be a source of fire specific data that can be integrated with the IAI schema developments. Figure 5-7, adapted from Liebich and Wix (2000), shows how fire engineering-related aspects can be integrated within the IAI architecture.

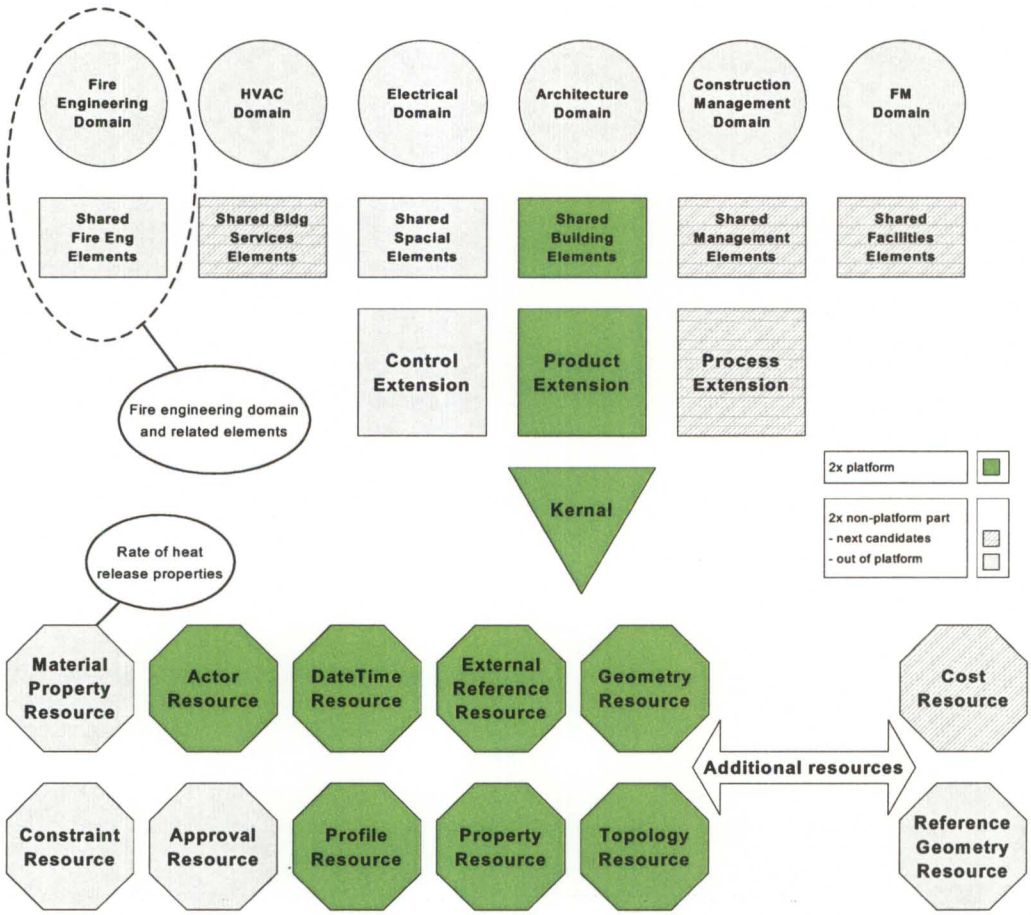


Figure 5-7. Integration of fire engineering with the IAI IFC release 2x architecture.

Integration with currently available fire engineering tools is already being undertaken. Appropriate fields in **FDMS** format data can be automatically mapped to the FireBaseXML schema by using a simple program. The ability to directly query a FireBaseXML database from the **BRANZFIRE** model (Wade, 1999) is discussed in detail in Section 6.4. Clearly the integration with a wider range of fire engineering tools may require access to program source codes.

5.6 Conclusions

The development of integrated computer tools means that there needs to be widespread agreement on the way in which data is exchanged. A set of requirements for a database of rate of heat release measurements have been identified and the use of XML as a means of storing and retrieving such data meets these requirements. Even if XML is not currently used by developers of fire models or those that supply data for

these models, it is important for fire engineering to keep up-to-date with emerging technologies. As XML data exchange becomes more widespread then fire engineers will necessarily begin to interact with those in related disciplines that have embraced the technology.

Currently intermediate transformations are necessary to take a rate of heat release curve from a FireBaseXML database and use it in a fire model. By making the structure of the database open and by using a standard format it is possible that future fire models can interrogate such a database from within, without the need for the intermediate step.

Additional comments:

The following text was not included in the original paper in order to make the document fit the publication length limitations.

5.7 Impact of data selection

The selection of design fires for design work or an appropriate fire growth curve for forensic / experimental validation work can be of significant importance. Providing the fire engineer with a database of rate of heat release curves raises the issues of how such a database can impact on the results obtained from simulations and what the choice of 'nearest match' items has on quality assurance.

In the design context, a generic fire growth curve which represents a reasonable worst case scenario is often selected such as the standard slow, medium, fast and ultra-fast fire curves provided in such documents as the USA's National Fire Alarm Code (NFPA, 1996). The use of these generic fire growth curves is not of such relevance in terms of quality assurance because of their wide acceptance by the fire engineering community.

For forensic / experimental validation work the selection could have a significant impact on the final results. In the absence of actual experimental measurements the fire engineer has to choose a fire growth they believe best represents the fire. Clearly the choice of 'nearest match' will depend on a subjective opinion and such an opinion may impact on the quality assurance aspects of a study where the final results are highly dependent on the input selection. A simulation study would likely also include some form of sensitivity analysis that would examine whether variability in the selected rate of heat release would significantly alter the simulation outcomes. The database aids the fire engineer in conducting a sensitivity analysis by presenting a collection of fire growth curves that might be appropriate alternatives to the 'nearest match'. In some instances data for only a single item of a particular type may be available and so the fire engineer may have a more limited opportunity for sensitivity analysis. Manually modifying the rate of heat release provided for the single item may be an option. However, the extent of the modification will depend on the judgement of the fire engineer with subsequent effects on the simulation outcomes.

5.8 Data requirements

In order to determine what should be the minimum data requirement of the database is, the fire curve parameters for several fire software tools had been previously reviewed by Spearpoint (1992b) and a summary of that review is given here.

Several forms of input data are used by **FPEtool** (Deal, 1995) to define a fire curve. If the rate of heat release of the fire is known then the fire is described by a constant heat of combustion (kJ/g) and the heat release at specified times (sec, kW). Alternatively, if a mass loss rate is known then the fire is described by a constant heat of combustion (kJ/g) and the mass loss at specified times (sec, g/s). Combinations of these parameters can also be used such as heat release and mass loss at specified times (kW, g/s); heat release and heat of combustion at specified times (kW, kJ/g) or mass loss and heat of combustion at specified times (g/s, kJ/g). The **ASET** model (Cooper, 1982) and the fire curve definition used in **ASKFRS** (Chitty et al., 1988) both use a constant heat of combustion (kJ/kg) and a set of rate of heat release data points (kW). The multiple fuels (**MULFUEL**) module in **HAZARD I** (Bukowski et al., 1989) requires a textual item description; heat of combustion (kJ/kg) and mass loss (kg) and species yields (kg/kg) for two to five fuels for selected times (s). Species yields are hydrogen, carbon monoxide (CO), carbon dioxide (CO2), Soot, hydrogen cyanide (HCN) and hydrogen chloride (HCl).

The **CFAST** software (Jones et al., 2000) allows storage of fire data obtained from a furniture calorimeter. Information includes text to describe the test item configuration and ignition source; the initial and final mass (g); ignition time (s); average heat of combustion (kJ/kg); average toxic gas yields (kg/kg) of carbon monoxide (CO), carbon dioxide (CO2), hydrogen chloride (HCl), hydrogen cyanide (HCN), water (H2O) and soot; the extinction area (m^2/kg). Data is also held for heat release rate (kW/m^2); mass loss (kg/s); CO yield (kg/kg) and CO2 yield (kg/kg) for ten selected times. When conducting a **CFAST** simulation, the following general parameters are required to specify the fire: heat of combustion (kJ/kg) and the oxygen (O2) limit (%) required for combustion. A series of time intervals (s) are required and then the following values can be specified for each time interval: pyrolysis mass loss (kg/s);

heat release rate (kJ/s); fire height (m); fire area (m^2); H/soot (fraction); CO/CO2 (fraction); soot/CO2 (fraction); HCN (fraction); HCl (fraction).

It can be seen from the information given above that the various fire engineering software tools allow for different forms of fire specification. However the common requirements across the tools are heat of combustion and the heat release rate and thus these would be seen to be the minimum requirements for a database.

5.9 Database units

Section 5.3.2 describes aspects of the database schema design and Appendix B.1 gives a complete listing of the schema. The unit types and restrictions currently defined in the schema are shown in Table 5-1.

Database element	Unit type	Unit restrictions
initial_mass	mass	kg, g
heat_of_combustion	available_energy	J/kg, kJ/kg
time	time	s
rhr	heat_flow_rate	kW

Table 5-1. FireBaseXML unit types and restrictions.

There is scope to both widen the unit restrictions available for the current unit types and append additional unit types to the list if new elements are added to the database. The current unit restrictions are limited to SI units and it would be possible to include Imperial units such as British Thermal Units (BTU) for 'heat_flow_rate' and pounds for 'mass'. These units are not as commonly used internationally as they were in the past and conversion tools could be written for individual extraction and insertion. Additional time units such as minutes and/or hours could also be included in an updated schema.

The review of the fire parameters used by various fire engineering software tools given in Section 5.8 identified a number of other data items that could be included in a revised FireBaseXML schema. These data could include product yields and mass loss

in particular and if the schema was updated to include such data then the unit types and restrictions are suggested in Table 5-2.

Database element	Unit type	Unit restrictions
mass_loss	mass_flow_rate	kg/s, g/s
CO	yield	kg/kg, g/g
CO2	yield	kg/kg, g/g
HCl	yield	kg/kg, g/g
HCN	yield	kg/kg, g/g
H2O	yield	kg/kg, g/g
soot	yield	kg/kg, g/g

Table 5-2. Proposed revised FireBaseXML unit types and restrictions.

5.10 Storing serial data in XML documents

5.10.1 Serial data

Serial data is typically list of numerical values that describe the change in the variable. Quite often this data is associated with another variable and a simple example is the change in temperature of some instrument with time. Potentially there may be several inter-related variables. Each variable will have its own associated units and potentially two variables that describe property may have different units. Consider the following table of data (Table 5-3) that has two temperature columns and related time column.

Time, t [s]	Instrument 1, T1 [°C]	Instrument 2, T2 [K]
0	20	293
1	22	298
2	28	299
3	30	305

Table 5-3. Example data set.

The XML document structure allows serial data to be stored in a variety of ways. It is not immediately obvious which storage format is the most appropriate and the following section discusses how such data could be stored in XML. Several methods

are examined with the advantages and disadvantages of each method investigated. In deciding how serial data is stored, the following two factors need to be considered:

- Compactness - the physical number of characters needed to specify the data and supporting tasks. This may be of particular significance where large volumes of data are delivered over a slow network connection.
- Flexibility - the ability to sort and manipulate the data. Consideration needs to be made as to the requirements for extraction and transformation. The transformation of the data might be based solely on using XSL or alternatively using a scripting language such as JavaScript or VBScript may need to be considered.

5.10.2 Storage method 1

Here data is stored in columns separated by semi-colons and each row separated by commas. Alternatively each row of data could be separated with semi-colons. The units for each data set are declared in the units attribute in the same order in which they appear in the element.

```
<table>
        <data units='s;C;K'>
                0, 1, 2, 3; 20, 22, 28, 30; 293, 298, 299, 305;
        </data>
</table>
```

This format for serial data is very compact but not very flexible. For example, it would take considerable effort in terms of coding to extract the time-temperature data set for Instrument 1. The extraction process would need to be able to infer the time and relevant temperature variables and then use some processing algorithm to find the associated values.

5.10.3 Storage method 2

This format, although less compact than the former, makes it significantly easier to extract the variables from the document. A <data> element holds a number of child elements that each stores a column of the original data. The child elements are give a name that reflects their data type and a units attribute is used to specify the unit type.

```
<table>
        <data>
            <time units='s'> 0, 1, 2, 3 </time>
            <temperature units='C'> 20, 22, 28, 30 </temperature>
            <temperature units='K'> 293, 298, 299, 305 </temperature>
        </data>
</table>
```

However both the method 1 and method 2 formats fail to relate the variables with the instruments in a form that explicitly associates a data set with an instrument. Instead, any extraction process would have to have some form of additional knowledge regarding which set of data relates to which instrument.

5.10.4 Storage method 3

This format now allows the variables associated with each instrument to be defined. Each instrument has its own <instrument> element with the name held in its attribute. A child <data> element is then used to hold the serial data using a similar format to method 1.

```
<table>
    <instrument name='T1'>
        <data units='s;C'> 0, 1, 2, 3; 20, 22, 28, 30 </data>
    </instrument>
    <instrument name='T2'>
        <data units='s;K'> 0, 1, 2, 3; 293, 298, 299, 305 </data>
    </instrument>
</table>
```

However it still exhibits some of the inflexibilities of method 1 such that it would take considerable effort in terms of coding to extract each time-temperature data set.

5.10.5 Storage method 4

Method 4 integrates methods 2 and 3 so as to allow instrument-variable associations and relatively simple extraction. The structure also uses attributes to explicitly declare both the data type and units. A <data> element is used to store the time data and then each instrument has its own <instrument> element with its name specified in the name attribute. The associated data is stored in a child <data> element with both the data type and units specified in appropriate attributes.

```
<table>
    <data type='time' units='s'>0, 1, 2, 3</data>
    <instrument name='T1'>
        <data type='temperature' units='C'> 20, 22, 28, 30 </data>
    </instrument>
    <instrument name='T2'>
        <data type='temperature' units='K'> 293, 298, 299, 305 </data>
    </instrument>
</table>
```

However this point the possibility of instruments having different timescales should be considered (i.e. durations and/or frequency) the above format does not allow for this.

5.10.6 Storage method 5

Method 5 expands on method 4 with the addition of the ability to assign different timescales to each entity. The <data> element used to store the time is repeated as a child element in each <instrument> element.

```
<table>
    <instrument name='T1'>
        <data type='time' units='s'>0, 1, 2, 3</data>
        <data type='temperature' units='C'> 20, 22, 28, 30 </data>
    </instrument>
    <instrument name='T2'>
        <data type='time' units='s'>0, 1, 2, 3</data>
        <data type='temperature' units='K'> 293, 298, 299, 305 </data>
    </instrument>
</table>
```

A complete instrument or all the instruments that are temperatures etc could now be extracted. by examining the type attribute. The format remains reasonably compact and XSL could be used to obtain the actual values by splitting the comma separated values (see Section 6.2.3.2 for an example on how this is achieved in XSL).

5.10.7 Storage method 6

Method 6 is that the most verbose format. It allows the extraction of individual values using XSL since each value has its own tag. However this format requires a significant volume of characters. For each value in the variable the overhead is 11 characters. This overhead may become an important consideration where a large database exists.

```
<table>
    <instrument name='T1'>
        <data type='time' units='s'>
            <val>0</val>
            <val>1</val>
            <val>2</val>
            <val>3</val>
        </data>
        <data type='temperature' units='C'>
            <val>20</val>
            <val>22</val>
            <val>28</val>
            <val>30</val>
        </data>
    </instrument>
    <instrument name='T2'>
        <data type='time' units='s'>
            <val>0</val>
            <val>1</val>
            <val>2</val>
            <val>3</val>
        </data>
        <data type='temperature' units='K'>
            <val>293</val>
            <val>298</val>
            <val>299</val>
            <val>305</val>
        </data>
    </instrument>
</table>
```

5.10.8 Storage method 7

This format collects the data together into time/measurement pairs. Units are specified in a similar form as shown in methods 1 and 3 and the data held in a series of <data> child elements associated with each <instrument> element. As with the methods 1 and 3, it is difficult to easily distinguish the units associated with each measurement.

```
<table>
    <instrument name='T1'>
        <data units='s;C'> 0, 20 </data>
        <data units='s;C'> 1, 22 </data>
        <data units='s;C'> 2, 28 </data>
        <data units='s;C'> 3, 30 </data>
    </instrument>
    <instrument name='T2'>
        <data units='s;K'> 0, 293 </data>
        <data units='s;K'> 1, 298 </data>
        <data units='s;K'> 2, 299 </data>
        <data units='s;K'> 3, 305 </data>
    </instrument>
</table>
```

5.10.9 Selection of storage method

In order to balance compactness and flexibility method 5 was selected to be the primary XML document structure for the database and document fragments. This storage method can be converted to other methods if required using XSL. The method allows for data sets that have different associated timescales which method 4 otherwise fails to address. Furthermore, a method that explicitly separates the specification of units, such as shown in methods 2, 4, 5 or 6, and uses of the type and units attributes, such as methods 4, 5 or 6, more easily allows for the automatic checking of restrictions as discussed in Section 5.3.2.

5.11 FireBaseXML database source publications

5.11.1 Cross-referencing publications

As described in Section 5.3.1, the FireBaseXML schema includes the ability to reference the original source publications from where the data was taken. It is import that this information be given and is correct so that a user of the database is able to refer back to the original work if needed.

Rather than the developer of a FireBaseXML database having to repeat the same publication details for multiple records, they can use the idref attribute to provide intra-document links to <reference> elements. Figure 5-8 illustrates how the idref attribute is used; the first record contains the complete <reference> element including an id attribute in line 3. In line 11, the <reference> element in the second record uses the idref attribute with the same value as the id attribute in the first record. Further records could use the same idref attribute if they are associated with the same original source publication.

```
1:  <item type="discrete" id="EASYF21">
2:      <attribute Armchair="Upholstered"/>
3:      <reference id="NBSIR-82-2604" type="Report" organisation="NIST/NBS">
4:          <title>Upholstered Furniture Heat Release Rates Measured with a
                Furniture Calorimeter</title>
5:          <document>NBSIR 82-2604</document>
6:          <authors>Babrauskas V, Lawson J R, Walton W D et al</authors>
7:      </reference>
        ...
8:  </item>
9:  <item type="discrete" id="EASYF22">
10:     <attribute Armchair="Upholstered"/>
11:     <reference idref="NBSIR-82-2604"/>
        ...
12: </item>
```

Figure 5-8. Example of using the idref attribute.

When using a cross-referenced <reference> the operation of the <link> element needs to be considered. A link points to a web address that might be a web-page or an Acrobat Portable Document Format document (PDF file). As such, the link may include a unique page number in an Acrobat document relating to a specific record. Thus a cross-referenced <reference> could be used to provide a unique <link> element. There are three ways to define a link and these are described here.

5.11.2 Basic 'in-reference' method

The <reference> and <link> elements are stored in a primary item record. For example, the <link> element below points to a PDF file at the URL http://www.brand.lth.se /bibl/3070.pdf with page 89 specified.

```
<item type="discrete" id="CAR1">
        <attribute Vehicle="Car"/>
        <reference type="Journal paper" id="FSJ-1994-MK">
                <title> Fire Behavior of a Burning Cars</title>
                <document>FSJ, Vol. 22, No. 3, 1994.</document>
                <authors>Mangs J, Keski-Rahkonen O</authors>
                <link type="pdf">
                        <URL>http://www.brand.lth.se/bibl/3070.pdf</URL>
                        <page>89</page>
                </link>
        </reference>
        [etc.]
</item>
```

5.11.3 Standard 'cross-reference' method

The link is taken from the cross-referenced <reference> element as shown here:

```
<item type="discrete" id="CAR2">
    <attribute Vehicle="Car"/>
    <reference idref="FSJ-1994-MK"/>
    [etc.]
</item>
```

Since this record uses the idref attribute to cross-link with the reference in the primary record it will also have the same link properties as the primary record. Thus, in the example, the link will be taken from the original reference, so that the URL would read http://www.brand.lth.se/bibl/3070.pdf#page=89 as it would for the primary record.

5.11.4 The 'cross-reference' method with unique link

In this case the author details etc. are taken from the cross-reference but a unique link is specified in the record that overrides that provided in the cross-reference.

```
<item type="discrete" id="CAR3">
        <attribute Vehicle="Car"/>
        <reference id="FSJ-1994-MK">
                <link type="pdf">
                        <URL>http://www.brand.lth.se/bibl/3070.pdf</URL>
                        <page>100</page>
                </link>
        </reference>
        [etc.]
</item>
```

Thus, the link given in the cross-reference is replaced with http://www.brand.lth.se/bibl/3070.pdf#page=100. To use this approach all of the <link> elements must be given in the <reference> element of the associated record.

5.12 The SelectFire application

5.12.1 Development

This section expands on Section 5.4.4 and details how **SelectFire** is structured and how it functions. **SelectFire** is a client application specifically for use with FireBaseXML heat release rate databases. The application allows searches of the database using generic categories to describe the characteristics of the records. Selected items can be viewed graphically and extracted via XSL transformations.

Additional third-party transformations can be made available to the application to extend the extraction capabilities.

The **SelectFire** application was originally based on a DOS program (known as **FireGen**) which was used to access a collection of rate of heat release data (Spearpoint, 1993a). The **SelectFire** application has significantly evolved from that early work and now runs under the Windows platform. Initial versions of the application were developed using Microsoft **Visual Basic 6.0** and the current version has been updated to Microsoft **Visual Basic .NET**. The latest version of **SelectFire** (version 2003.1) detailed in the following sub-sections has several new features added compared to the release described in Section 5.4.4 and illustrated in Figure 5-6.

5.12.2 Database operations

When **SelectFire** is run it opens the last FireBaseXML database accessed and then displays the Search window (Figure 5-9). New databases can be either accessed from a local disk of from a web server. Once a new database has been selected, the data description is added to the list of available databases. A database homepage can be viewed by clicking on the Database home button in the *Search* window. A web browser window will show the homepage for the database which is generated from the original source XML database file (see Section 6.2).

Searches can be carried out once a database has been successfully opened. The drop down lists can be used to select the appropriate Groups and Categories and list of record matches are displayed in the *Results* list.

The rate of heat release curve and associated information can be viewed by the user selecting an item from the *Results* list and clicking the *View* button to open the *View* window. A selected item record can be extracted by using the *Extract* button which then allows the user to apply a conversion transformation to create an output file in a new format.

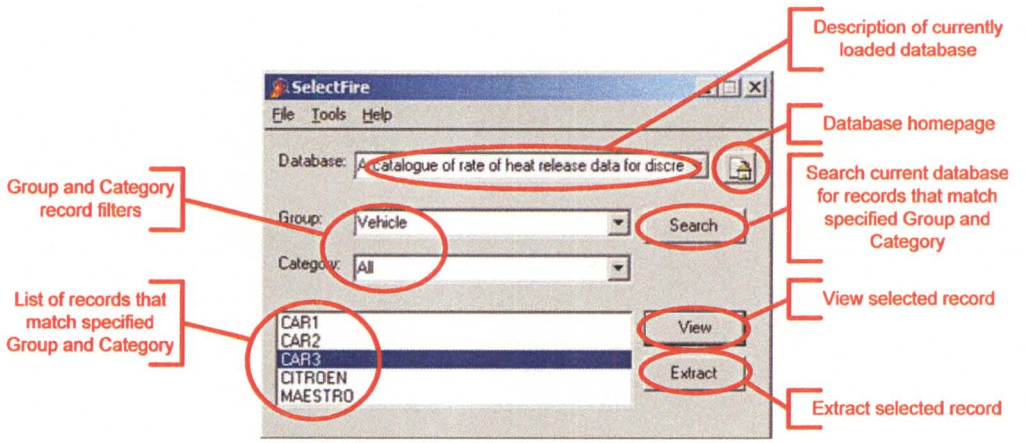


Figure 5-9. The **SelectFire** *Search Window.*

The *View* window (Figure 5-10) contains a plot of the rate of heat release curve and test details. The units displayed on the axes are those in which the data is stored in the database.

The *View* window also allows the user to print the current item and the *Extract* button has the same function as the *Extract* button in the *Search* Window. Additional buttons will appear in the *View* window if a record includes a link to a web page or an Adobe Acrobat document. Clicking on the button will automatically open the web page or open the Acrobat document at the appropriate page.

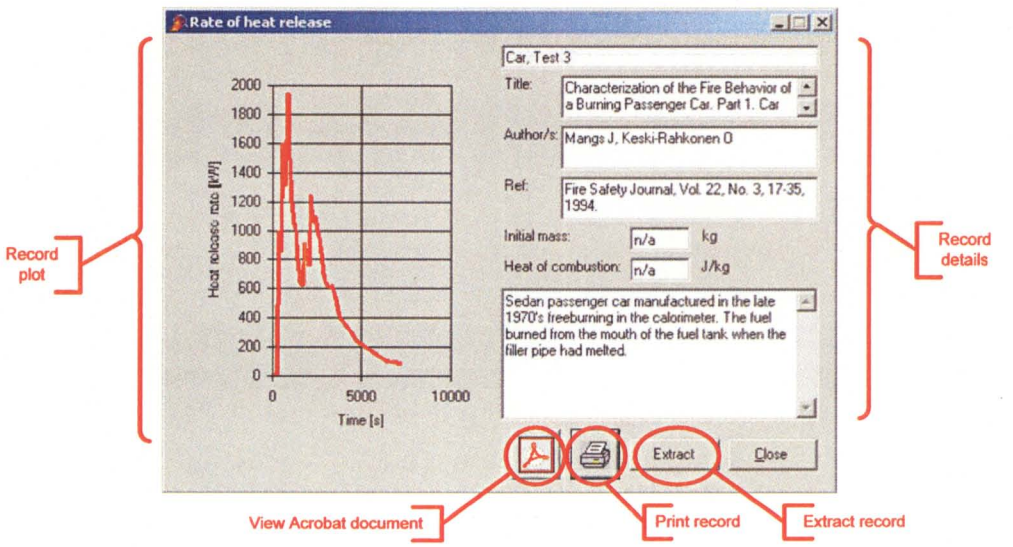


Figure 5-10. The **SelectFire** *View* window.

5.12.3 Conversion transformations

Transformations that can be used to convert a record to an output file in a new format can be either accessed from a local disk or from a web server. The structure of a conversion transformation and the types of conversions available are detailed in Section 6.3.3.

Transformations can be viewed, added or deleted using the *Transformation Management* window (Figure 5-11). Viewing the transformation properties (source document, author, version) can be toggled on or off by clicking the *Properties* button and these properties are extracted from the FireBaseXML Conversion schema elements (see Section 6.3.2).

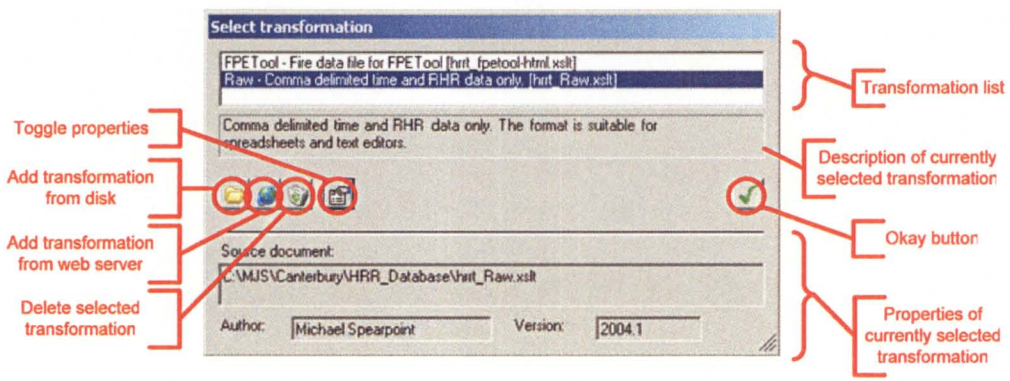


Figure 5-11. The **SelectFire** *Transformation Management* window.

When a record is extracted from a database the *Select Transformation* dialogue box appears listing the available transformations. Each transformation is indicated by its description and filename. If a transformation cannot be accessed it will be shown as `**Unavailable**` in the list. Transformations may not be available because the server it resides on is not online or the transformation no longer exists.

In certain cases a selected transformation cannot be completed without further details being supplied by the user. In that case, a supplementary dialogue box appears and the user needs to respond with the relevant information (see Section 6.3.3.3 and

Section 6.3.3.4). When a record is extracted, the rate of heat release data can be output at user specified regular time steps using an interpolation method.

5.12.4 Miscellaneous information

Installation of the **SelectFire** application is accomplished using its associated setup program, Setup.msi. Some of the lists of known databases and conversion transformations used by **SelectFire** are stored in its configuration file. The configuration file is an XML document which follows the schema shown in Figure 5-12.

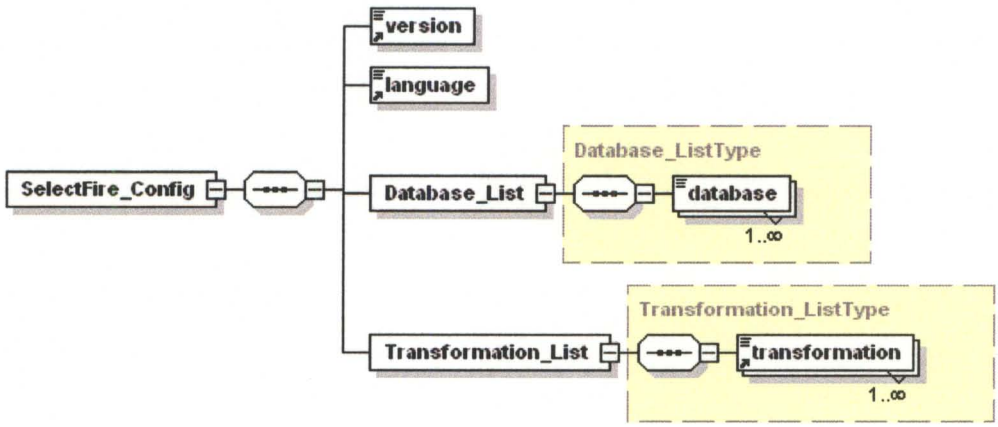


Figure 5-12. The **SelectFire** configuration schema.

SelectFire also includes an associated set of online help documents which were created as a series of web pages (Figure 5-13) and then amalgamated into a Help file. This Help file (Figure 5-14) can be opened from the *Search* window.

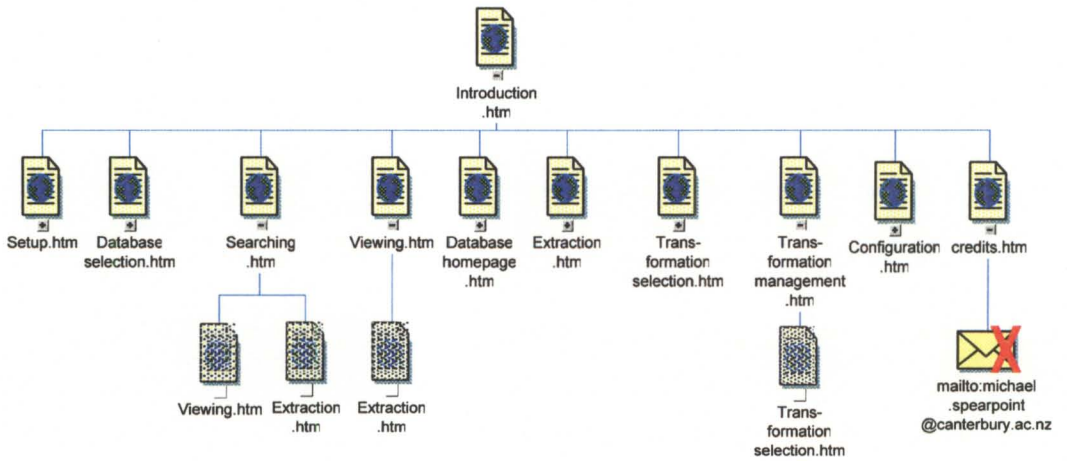


Figure 5-13. The **SelectFire** help file structure.

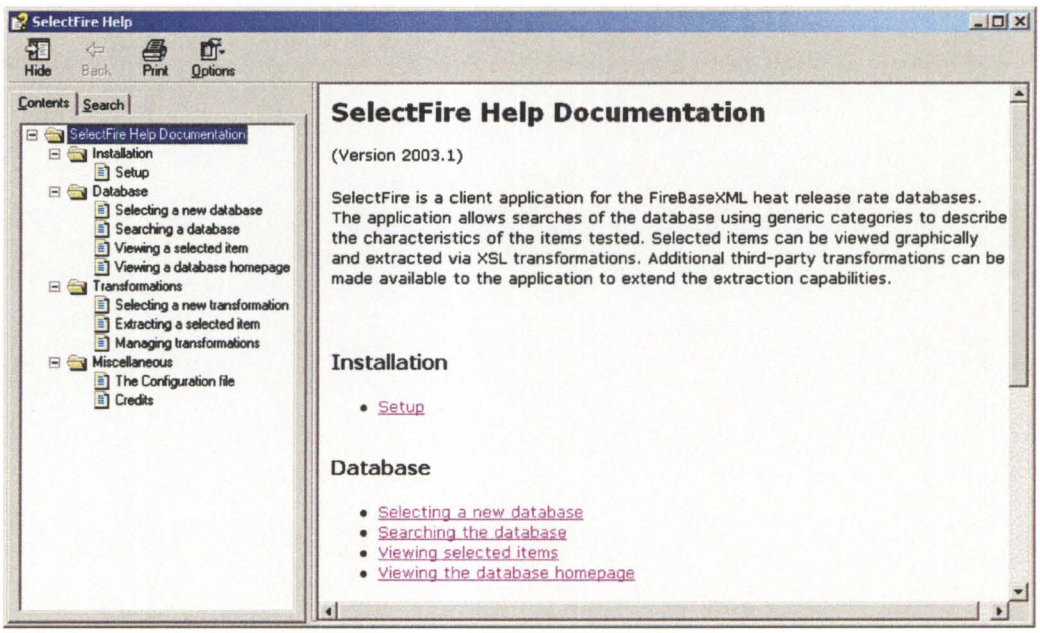


Figure 5-14. The **SelectFire** *Help* window.

Chapter 6:
INTERFACES TO THE FIREBASEXML DATABASE

6.1 Chapter introduction

In this chapter the use XSL transformations and the self-documenting ability of XML documents to provide information and data conversion control to a user is examined. In particular, the self-documenting functions to provide an interface to the FireBaseXML documents are used. This demonstrates how the database can be viewed using freely available software rather than needing specifically developed code.

The chapter then goes on to detail how XSL is used to allow the user to convert a selected database record into an alternative format. The converted record can then be viewed or processed by another software tool such as a spreadsheet or a fire simulation model. Finally, a specific use of an XSL transformation that allows direct interface to commercially available fire simulation software tool is shown.

6.2 Web-based interface to FireBaseXML databases

6.2.1 Header information

Figure 6-1 shows a fragment of a FireBaseXML database document. The individual records are stored in the <item> elements (lines 8 to 9) and these are described in more detail in Section 5.3.1.

```
1:  <?xml version="1.0" encoding="UTF-8"?>
2:  <?xml-stylesheet type="text/xsl" href="viewDatabase.xslt" ?>
3:  <FireBaseXML xmlns:hrrs="FireBaseXML.xsd"
        xmlns:xsi=http://www.w3.org/2000/10/XMLSchema-instance
        xsi:noNamespaceSchemaLocation="FireBaseXML.xsd">
4:      <version>1.31</version>
5:      <date>2003-04-16</date>
6:      <author email="m.spearpoint@civil.canterbury.ac.nz">
            Michael Spearpoint</author>
7:      <description> A catalogue of rate of heat release data for discrete fuel
            packages</description>
8:          <item>

                [etc.]

9:          </item>
10: </FireBaseXML>
```

Figure 6-1. FireBaseXML database document header.

The document header information is declared in lines 1 to 7. These elements in the header can be used to provide an interface to the FireBaseXML documents via web pages and be used by any other client software to obtain details regarding the FireBaseXML database.

When a FireBaseXML document viewed in a browser, it is automatically transformed into a webpage view of the document. This is declared in line 2 of the header where the XSL script that performs the transformation is called viewDatabase.xslt. The webpage generated by the viewDatabase.xslt transformation (Figure 6-2) shows the header information given in lines 4 to 7.

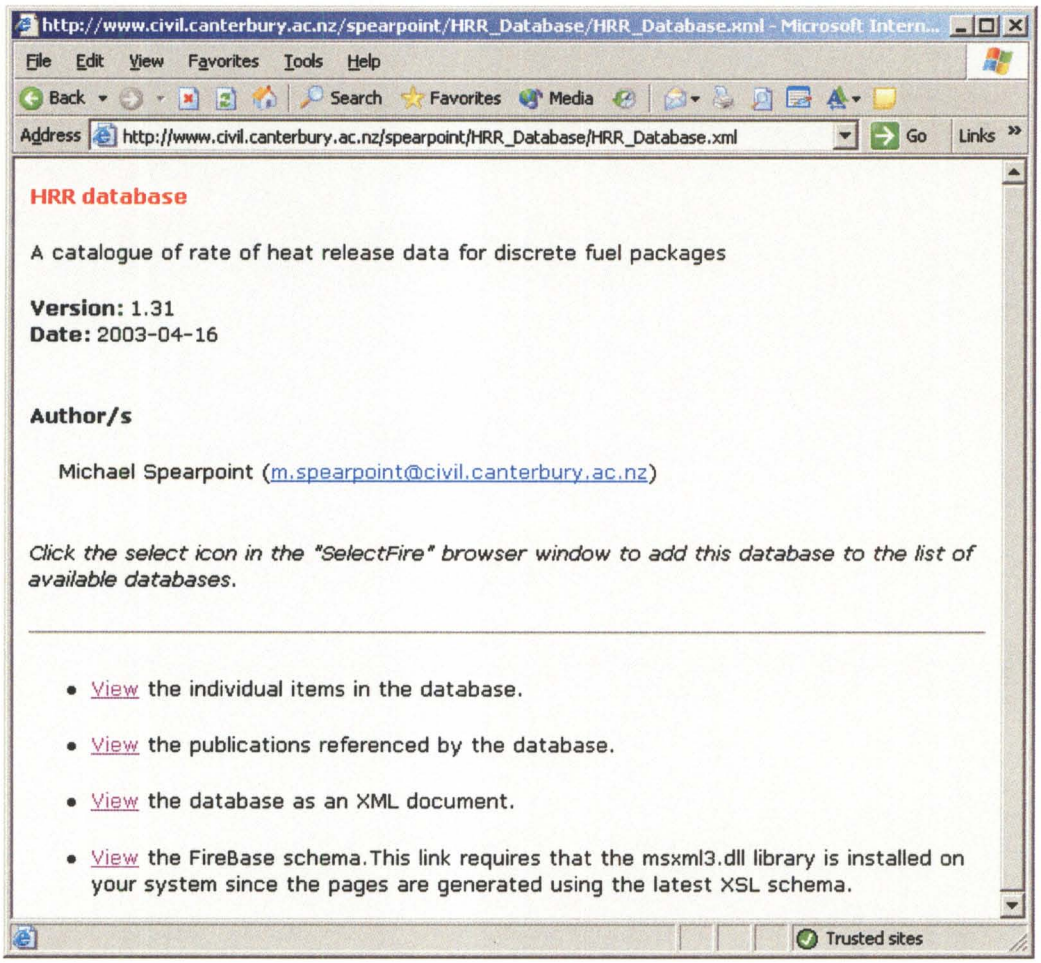


Figure 6-2. Default webpage view of a FireBaseXML database generated by the viewDatabase.xslt transformation.

In addition, the webpage gives the opportunity for the user to get further details about the database. This includes the ability to obtain a list of each record in the database, a complete list of each reference used in the database, a view of the source XML document and a navigable view of the FireBaseXML schema. Each of these sub-views of the database is provided as a webpage generated using XSL transformations. The process and the transformations required are summarised in Figure 6-3.

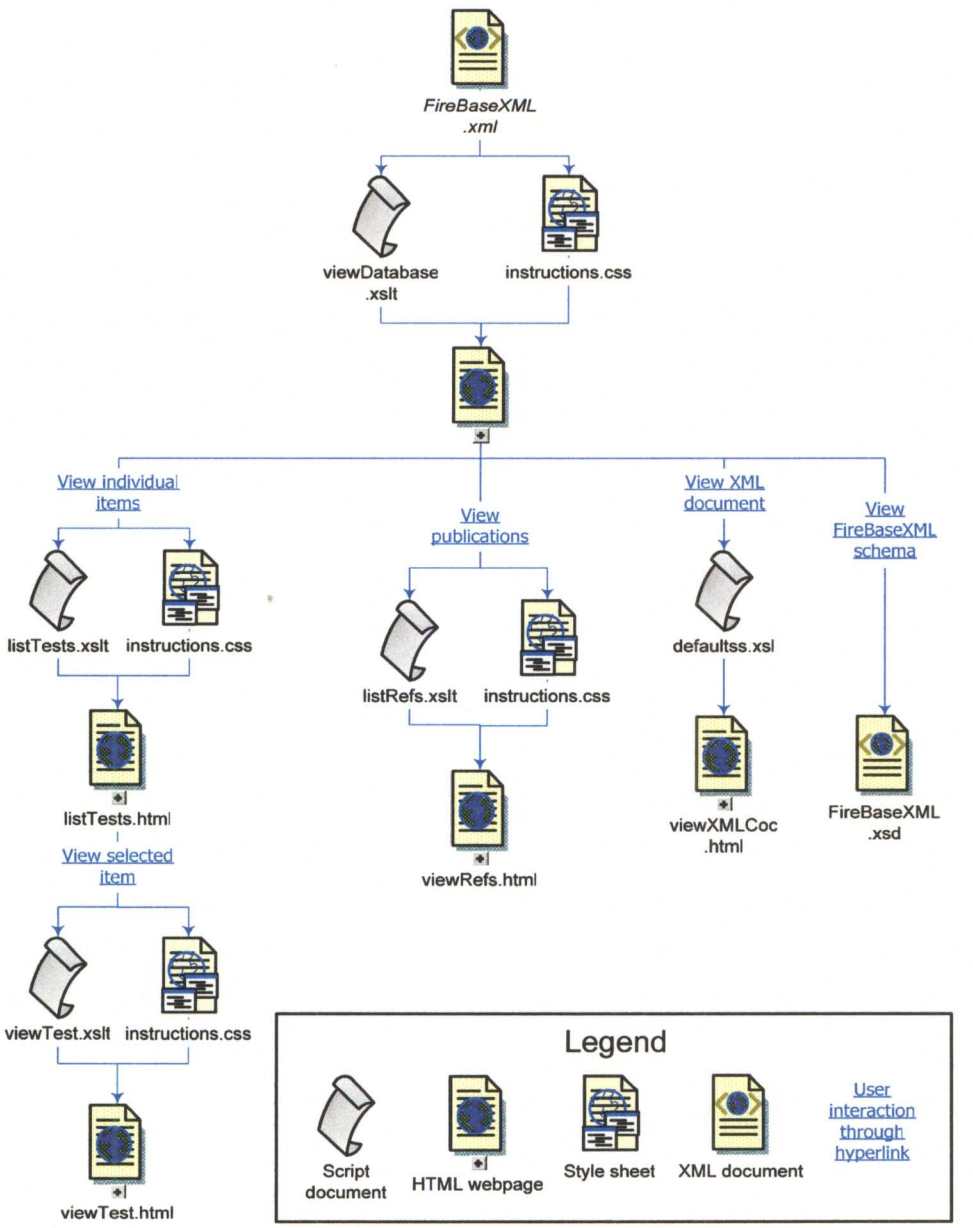


Figure 6-3. FireBaseXML database webpage views, associated transformations and style sheets.

Details of the techniques that have been employed to construct the sub-view web pages are given in detail below.

6.2.2 Listing test records

The listing of test records in a FireBaseXML database is given in the listTests.html webpage using the associated listTests.xslt transformation. The cascading style sheet for the listTests.html webpage is specified in the instructions.css file.

Figure 6-4 shows the listTests.html (version 2003.1) script and Figure 6-5 shows the listTests.xslt (version 2003.1) script. The listTests.html script is a mixture of HTML and JavaScript code (Goodman, 2001) and the listTests.xslt script uses HTML integrated into XSL commands. The result of these two scripts is illustrated in Figure 6-6.

```
1:  <html>
2:      <head>
3:          <title>Database item list</title>
4:          <link rel="stylesheet" type="text/css" href="instructions.css" />
5:      </head>

        <!-- Declare database and stylesheet -->
6:      <XML id="source"></XML>
7:      <XML id="style" src="listTests.xslt"></XML>


        <!-- Perform transformation -->
8:      <SCRIPT FOR="window" EVENT="onload" language="jscript">

            // Get the search component of the calling document and split by '='s
9:          list = document.location.search.split("=");
10:         path = list[1];

            // Pass the database document to the stylesheet
11:         with (style.selectSingleNode("//*[@name='databaseName']/text()")) {
12:             text = path;
13:             }

            // Load database and transform
14:         with (source) {
15:             async = false;
16:             load(path);
17:             }
18:         htmlObj.innerHTML = source.transformNode(style.XMLDocument);
```

```
19:     </SCRIPT>

20:     <body>
21:         <h1>Database item list</h1>
22:         <div id="htmlObj"></div>
23:         <hr align="center"/>
24:     </body>
25: </html>
```

Figure 6-4. Source listing for listTests.html.

```
1:  <?xml version="1.0" encoding="UTF-8"?>
2:  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

        <!-- Declare parameter that holds the source database name which is populated
             by the calling listTests.html file -->
3:      <xsl:param name="databaseName">dummy</xsl:param>

4:      <xsl:template match="/">

5:          <table>

            <!-- Search for each <item> element -->
6:          <xsl:for-each select=".//item">
7:              <tr>
8:              <xsl:apply-templates select="@id"/>
9:              <xsl:apply-templates select="description/summary"/>
10:             </tr>
11:         </xsl:for-each>

12:         </table>

13:     </xsl:template>


        <!-- id attribute template -->
14:     <xsl:template match='@id'>
15:         <td><a>

            <!-- Create href which includes the source database name and the name of
                 the test item -->
16:         <xsl:attribute name="href">viewTest.html?document=
17:         <xsl:value-of select="$databaseName"/>
18:         &amp;test=<xsl:value-of select="."/>
19:         </xsl:attribute>

20:         <xsl:value-of select="."/>

21:         </a></td>
22:     </xsl:template>
```

```
        <!-- <summary> element template -->
23:     <xsl:template match='summary'>
24:         <td><xsl:value-of select="."/></td>
25:     </xsl:template>

26: </xsl:stylesheet>
```

Figure 6-5. Source listing for listTests.xslt.

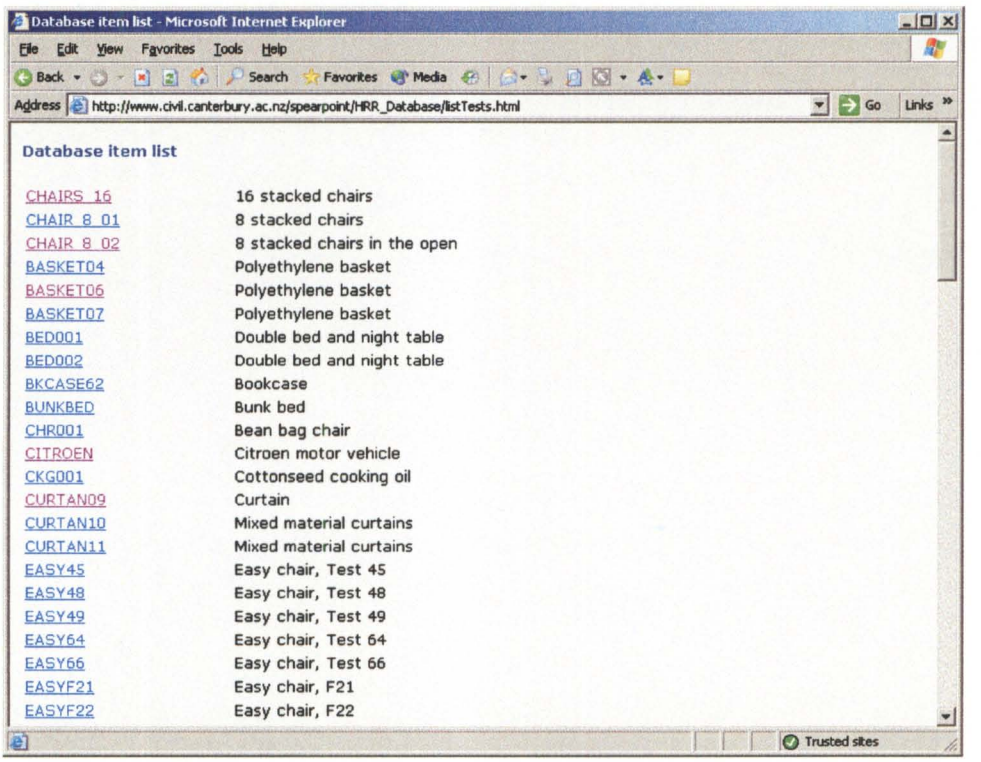


Figure 6-6. Webpage view of individual records in a FireBaseXML database generated by the listTests.html and listTests.xslt scripts.

6.2.2.1 The listTests.html *script*

Line 6 of listTests.html creates an XML document ready for loading the FireBaseXML document and line 7 declares the associated listTests.xslt transformation.

When the listTests.html document is opened the code between lines 8 and 19 is executed. The parent document that calls listTests.html passes the name of the FireBaseXML database in the URL in the form

```
listTests.html?document=Database_path.
```

Line 9 splits the URL about the equals sign so that the `Database_path` can be found in line 10.

In order to pass the actual path of the FireBaseXML database to the listTests.xslt code, line 11 in listTests.html searches for the <xsl:param> element with the attribute name="databaseName" in the XSL document (Line 3 in Figure 6-5). Line 12 then replaces the dummy name for the <xsl:param> element with the actual path of the FireBaseXML database.

Lines 14 to 17 then load the FireBaseXML database and line 18 applies the listTests.xslt transformation to the database. The result of this transformation is displayed as a part of the final webpage in line 22. The remainder of the final page are specified in lines 20 – 21 and 23 – 24 with the cascading style sheet for the webpage specified in line 4.

6.2.2.2 The **listTests.xslt** *script*

Lines 4 to 13 specify the root template that creates an HTML table of each individual record in the database. Lines 8 and 9 call the templates for the record id attribute and the record <summary> and <description> elements respectively.

Lines 14 to 22 are the template for the record id attribute. Lines 16 to 19 create a hyperlink to a record in which the URL gives the source FireBaseXML database path and the test id name in the form

```
viewTest.html?document=Database_path&test=Record_id
```

where the database path is obtained from the contents of the <xsl:param> element with the attribute name="databaseName".

Lines 23 to 25 are the description/summary template. The template simply displays the summary text associated with the record.

6.2.3 Viewing an individual test

Once the list of records has been generated, it is possible to then select an individual test and view its details. When the user selects a test record the viewTest.html webpage is accessed, though the hyperlink created by the listTests.xslt script, along with its associated viewTest.xslt transformation. Complete listings of the scripts are given in Appendix C.1 and Appendix C.2 respectively. An example of a result of these two scripts is illustrated in Figure 6-7. The structure of a FireBaseXML item record is shown in the schema diagram (Figure 5-2).

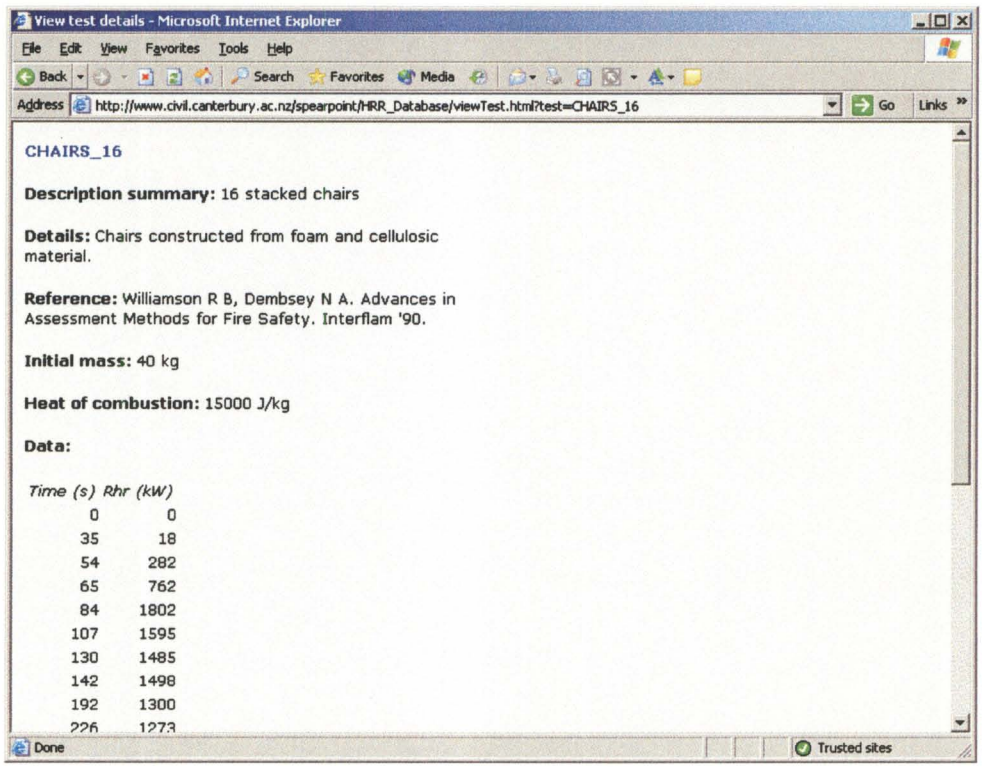


Figure 6-7. Webpage view of a selected individual record in a FireBaseXML database generated by the viewTest.html and viewTest.xslt scripts.

6.2.3.1 The viewTest.html *script*

The viewTest.html script performs in a similar way to the listTests.html script. However, since both the database path and test name are passed in the URL they need to be separated out of the URL during the processing of the viewTest.html script. Line

10 splits the URL about the '&' sign so that first element in the list array is `document=Database_path` and the second element `test=Record_id`. Lines 11 to 16 identify and load the database. Lines 17 and 18 identify the individual test id and construct a query sting that is used to select the appropriate XML node in line 19. Finally, line 20 applies the viewTest.xslt transformation to the XML node.

6.2.3.2 The viewTest.xslt script

The viewTest.xslt script transforms a selected database record to HTML. The majority of the script simply identifies information stored in the record and formats for the webpage. Templates for the <description> (lines 20 to 23), <initial_mass> (lines 60 to 65) and <heat_of_combustion> (lines 66 to 71) elements in the FireBaseXML database are examples of simple formatting.

For the <reference> element, the script in lines 30 to 59 needs to determine whether the reference details are associated with the currently selected FireBaseXML record or through an idref attribute. Templates are then used to format the <authors>, <title> and <document> elements.

The most complex part of the transformation is for the <data> elements. In the FireBaseXML database, the rate of heat release data is held in CSV format (see Section 5.3.2). For the webpage view, this information is displayed as a two-column table and so the CSV data needs to be broken into its component values. The viewTest.xslt script achieves this solely using XSL commands.

Lines 73 and 74 of the <data> template find the <time> and <rhr> elements in the record and populate two XSL variables. Lines 76 to 81 construct the heading of the webpage table. Lines 82 to 85 call the decompose template and passes the <time> and <rhr> element contents as parameters.

The decompose template breaks down the CSV lists. The template checks that the <time> data parameter string contains a comma character in line 94. If it does, the template uses the 'substring-after' and 'substring-before' functions to read the next value in the lists and truncates to new CSV data strings appropriately (lines 95 to 98). The template then checks that the truncated <time> data string has more than zero

characters in line 99. If there are more than zero characters then the template calls itself, passing the truncated data strings as parameters. Thus, the template iteratively decomposes the CSV list. When the template reaches the last value in the CSV lists there will not be any commas remaining. In that case, the last value in the CSV lists has been reached and the template need only output that last value (lines 106 to 109).

The viewTest.xslt (version 2004.1) script transformation process is illustrated in Figure 6-8. Each rectangle on the diagram represents a block of code associated with a specified template, e.g. the FireBaseXML rectangle in Figure 6-8 corresponds to lines 6 – 8 in Appendix C.2. The blocks are colour-coded to differentiate between templates related directly to elements and attributes associated with the FireBaseXML schema (yellow and cyan backgrounds respectively) and the decompose template that is specific to the viewTest.xslt script (orange background). Transfer between templates is shown using the arrowed lines and the decision process necessary in the <reference> element is indicated appropriately. The iterative process used by the decompose template is shown by a transfer arrow leading back to the template block.

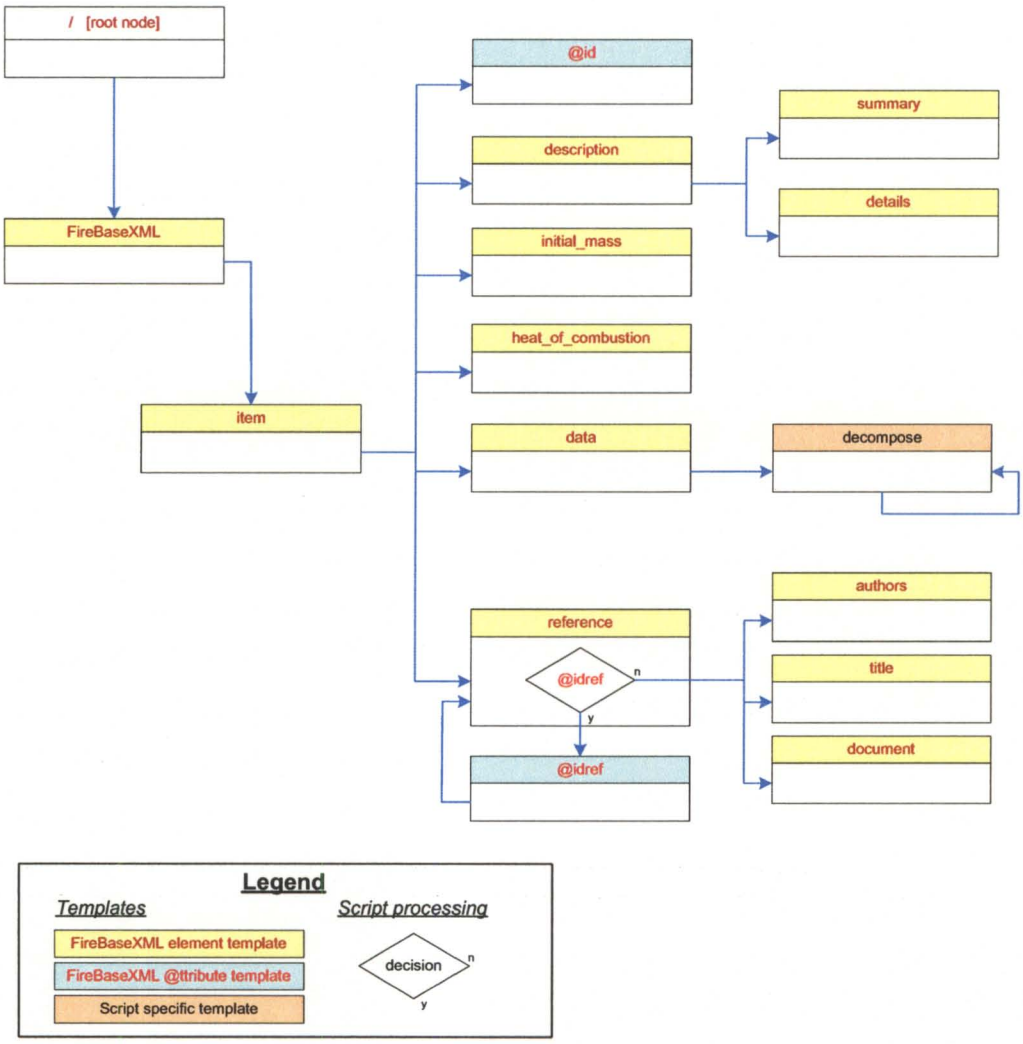


Figure 6-8. Transformation process diagram for viewTest.xslt.

6.2.4 Listing the source publications

The source publications referenced by the database can be made available to the user by the listRefs.html webpage using the associated listRefs.xslt transformation and the instructions.css style sheet. The listRefs.xslt transformation identifies all complete <reference> elements and ignores any <reference> elements that have an idref attribute (see Section 5.11). If a single publication is cited with a complete <reference> element more than once it will appear more than once in the webpage output. The listRefs.html and listRefs.xslt scripts work very similarly to the listTests.html and listTests.xslt scripts and so are not detailed further here.

6.2.5 Viewing the source XML document

Since the default action when a FireBaseXML database is opened is to execute viewDatabase.xslt transformation, a separate script named viewDatabaseXML.html is used to create a view of the source XML document as shown in Figure 6-9. The script uses the Microsoft Internet Explorer default style sheet defaultss.xsl and a collection of HTML style elements to create the webpage view of the source XML document. As with the listRefs.html script, the viewDatabaseXML.html script works very similarly to the listTests.html script and so is not detailed here.

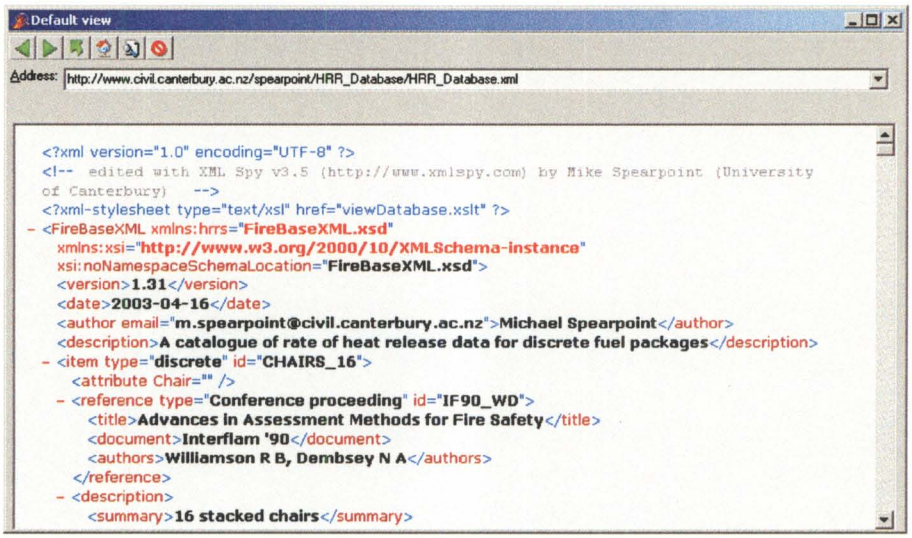


Figure 6-9. View of the source XML of a FireBaseXML document.

6.2.6 Viewing the FireBaseXML schema

As noted in Section 6.2, the user is able to obtain a navigable view of the FireBaseXML.xsd schema from the webpage generated by the viewDatabase.xslt transformation. Since schema documents are also valid XML documents a default transformation can be assigned to the schema document in a similar way as described above for a FireBaseXML document.

Two transformations were developed for generating the schema webpage view: a generic set of templates for any valid schema document and a set of templates specific to the FireBaseXML.xsd schema. A number of additional scripts were then required to

generate views of the schema elements. Figure 6-10 shows how these various scripts are related.

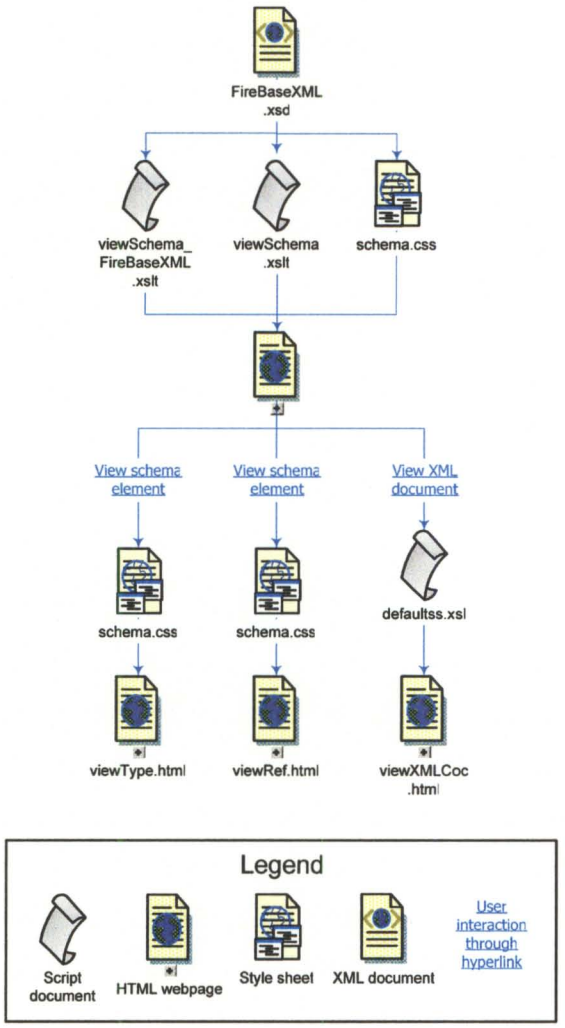


Figure 6-10. FireBaseXML schema webpage views, associated transformations and style sheets.

To generate the webpage view of the schema, the viewSchema_FireBaseXML.xslt, version 2004.1 (Figure 6-11) script is processed and this opens a more generic transformation viewSchema.xslt by the use of the <xsl:include href="viewSchema.xslt"/> statement in line 7.

```
1:  <?xml version="1.0" encoding="UTF-8"?>
2:  <?xml-stylesheet type="text/xsl" href="documentation.xslt"?>
3:  <xsl:stylesheet exclude-result-prefixes='xsd' version="1.0"
```

```
4:      xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
5:      xmlns:xsd="http://www.w3.org/2001/XMLSchema"
6:      <xsl:output indent='no' method='xml' />
7:      <xsl:include href="viewSchema.xslt"/>

        <!--
*************************************************************************************
*
                'name' attribute -->
8:      <xsl:template match="@name" mode="link" name="nameMatch">
9:          <li/>
10:             <span class="blu">&lt;</span>
11:             <a>
12:             <xsl:attribute name="target">Element viewer</xsl:attribute>
13:             <xsl:attribute
                    name="href">viewType.html?schema=FireBaseXML?element=
                    <xsl:value-of select="."/>Type</xsl:attribute>
14:             <span class="mrn"><xsl:value-of select="."/></span>
15:             </a>
16:             <span class="blu">&gt;</span>
17:         </xsl:template>


            <!--
*************************************************************************************
                'ref' attribute -->
18:         <xsl:template match="@ref" mode="link" name='refMatch'>
19:             <li/>
20:             <span class="blu">&lt;</span>
21:             <a>
22:             <xsl:attribute name="target">Element viewer</xsl:attribute>
23:             <xsl:attribute
                    name="href">viewRef.html?schema=FireBaseXML?element=
                    <xsl:value-of select="."/></xsl:attribute>
24:             <span class="mrn"><xsl:value-of select="."/></span>
25:             </a>
26:             <span class="blu">&gt;</span>
27:         </xsl:template>

            <!--
*************************************************************************************
                <xsd:schema> element -->
28:         <xsl:template match="xsd:schema">
29:             <html>
30:                 <head>
31:                     <title>Schema view</title>
32:                     <link rel="stylesheet" type="text/css" href="schema.css"/>
33:                 </head>

34:                 <body>
                    <!-- Find the only element with a complexType child -->
35:                     <xsl:for-each select="xsd:element">
36:                         <xsl:apply-templates select="xsd:complexType" mode="main"/>
```

```
37:                     </xsl:for-each>
38:                     <hr/>
39:                     <p><a href="viewXMLDoc.html?document=FireBaseXML.xsd">
                            View</a> schema as an XML document</p>
40:                 </body>

41:             </html>
42:         </xsl:template>

43:     </xsl:stylesheet>
```

Figure 6-11. Source listing for viewSchema_FireBaseXML.xslt.

Lines 28 – 42 includes the HTML tags that create the webpage and builds the tree of schema elements from the topmost <xsd:element> that has an <xsd:complexType> child element. Line 39 allows the user to open the source XML schema document in a browser window similar to the method described in Section 6.2.5. Lines 8 – 17 and 18 – 27 are where the additional viewType.html and viewRef.html scripts are executed in response to a user clicking on a schema element in the tree.

The processing of an XML schema document using the transformations is a relatively complex task due to the iterative nature of the schema structure. Rather than give details of the exact working of the transformation templates, a diagrammatic representation of the process is shown in Figure 6-12.

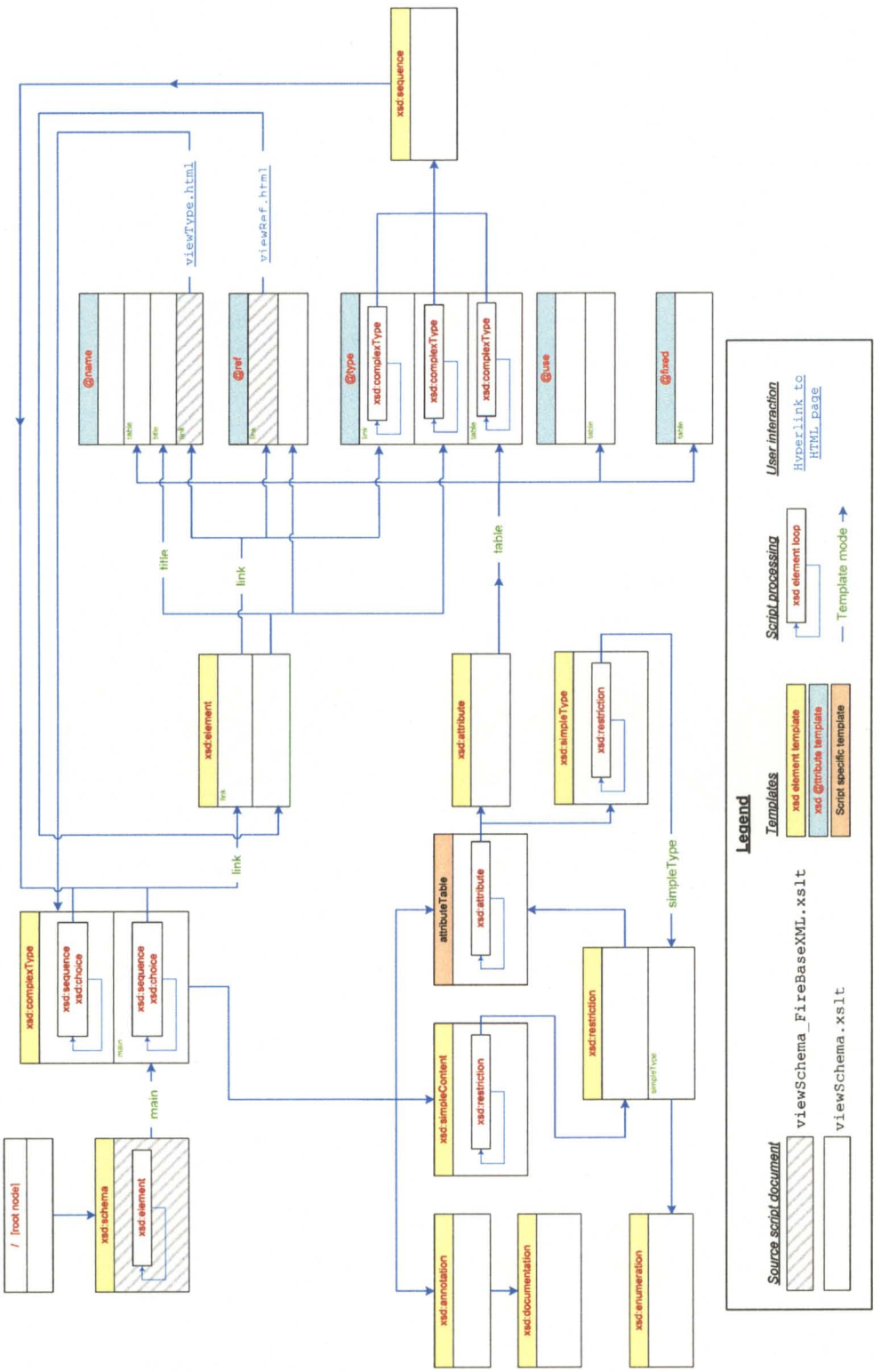


Figure 6-12. Transformation process diagram for viewSchema.xslt and viewSchema_FireBaseXML.xslt.

Figure 6-12 is similar to Figure 6-8 where each rectangle on the diagram represents a block of code associated with a specified template. The diagram identifies <xsl:for-each> loops with the selection element given in the loop box. For example, the xsd:element loop in the xsd:schema template rectangle shown in Figure 6-12 represents the following three lines of script:

```
1: <xsl:for-each select="xsd:element">
2:     <xsl:apply-templates select="xsd:complexType" mode="main"/>
3: </xsl:for-each>
```

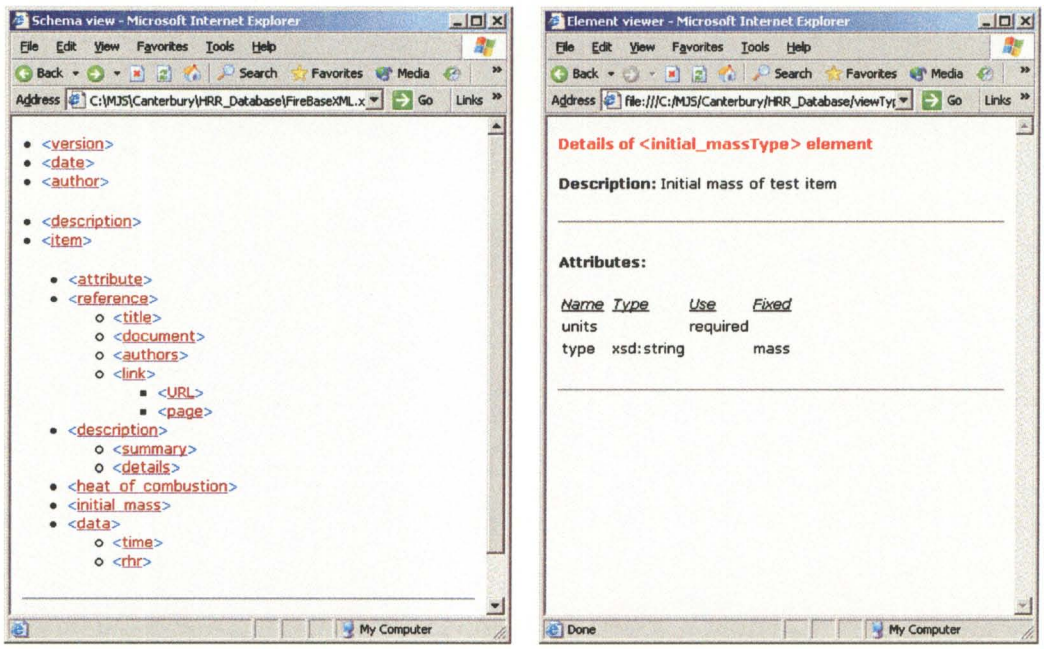
Transfer between templates is shown using the arrowed lines with mode names indicated where appropriate. Where there are several modes available to a template then each mode is indicated by one or more rectangles under the template name with the mode name given in the top left-hand corner. For example, the xsd:complexType template has two modes; the default mode and the 'main' mode symbolised by two rectangles below the template name. These two rectangles represent the following lines of script:

```
   <!-- mode = 'main' -->
1: <xsl:template match="xsd:complexType" mode="main">
       ...
2: </xsl:template>

   <!-- no mode -->
3: <xsl:template match="xsd:complexType">
       ...
4: </xsl:template>
```

The 'main' mode template script is entered from the xsd:element loop in the xsd:schema template as discussed above and the default template script is entered from elsewhere.

Figure 6-12 shows where the user is able to interact with the script processing through the use of hyperlinks. Finally, the diagram shows in which source document (i.e. viewSchema.xslt or viewSchema_ FireBaseXML.xslt) each template is to be found by using shading in specific rectangles. An example of the webpage output from the two scripts that enable the user to view a FireBaseXML schema is shown in Figure 6-13.



(a) (b)

Figure 6-13. Webpage view of (a) the FireBaseXML schema and (b) a FireBaseXML schema element generated by the viewSchema.xslt and viewSchema_FireBaseXML.xslt scripts.

6.3 Transforming a selected FireBaseXML record

6.3.1 Transformation process

In Section 5.4.3 it has been mentioned that XSL transformations were created allowing the conversion of a FireBaseXML record into an alternative format. In this section the range of conversions available at the time of writing are detailed and a description of how they operate provided.

The XSL transformation used to view a test record in a webpage (Appendix C.2) is similar to the XSL transformations that perform these conversions. A record is extracted from a FireBaseXML database and an XSL transformation is applied to that record. The result of the transformation is a new document formatted in accordance with the transformation script. Several FireBaseXML record conversion transformations have been created and each conforms to a specific schema for such transformations.

6.3.2 The FireBaseXML Conversion schema

The FireBaseXML Conversion schema (Figure 6-14) defines the structure of elements in a FireBaseXML record conversion transformation document. The elements are used to

- Give the name, author and version of the conversion.
- Provide a text description of the conversion document.
- Specify the file extensions that are applicable to the output result of the conversion.

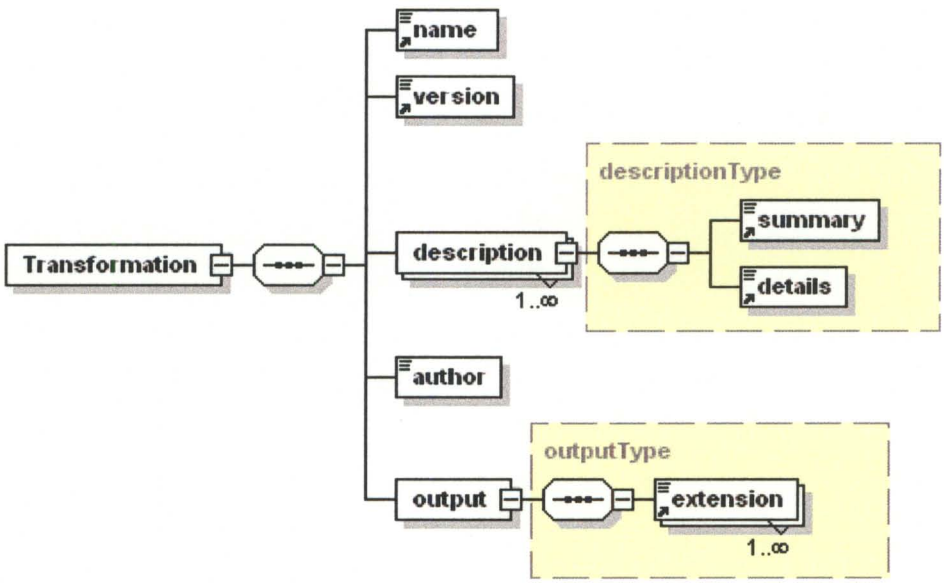


Figure 6-14. The FireBaseXML Conversion schema.

The elements can be used by an external program in order to provide a user detail of the conversion document such as shown in the **SelectFire** application (see Section 5.12.3). Just as with a FireBaseXML database document, described in Section 6.2, a default transformation can be specified for the FireBaseXML conversion documents so that they can be viewed as a webpage. The viewConversion.xslt (version 2004.1) transformation (Figure 6-15) accesses the FireBaseXML Conversion schema elements and generates a webpage (Figure 6-16).

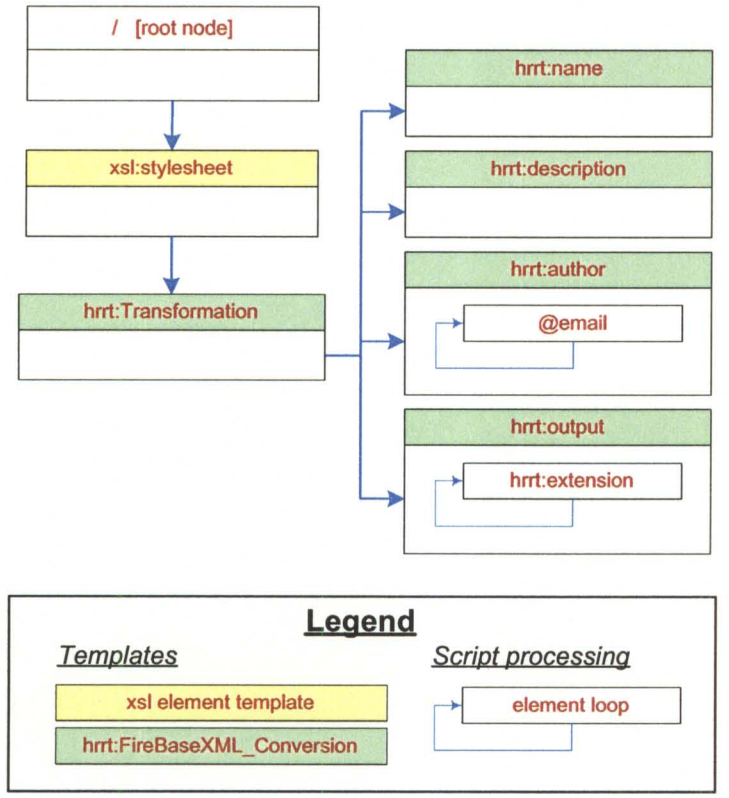


Figure 6-15. Transformation process diagram for viewConversion.xslt.

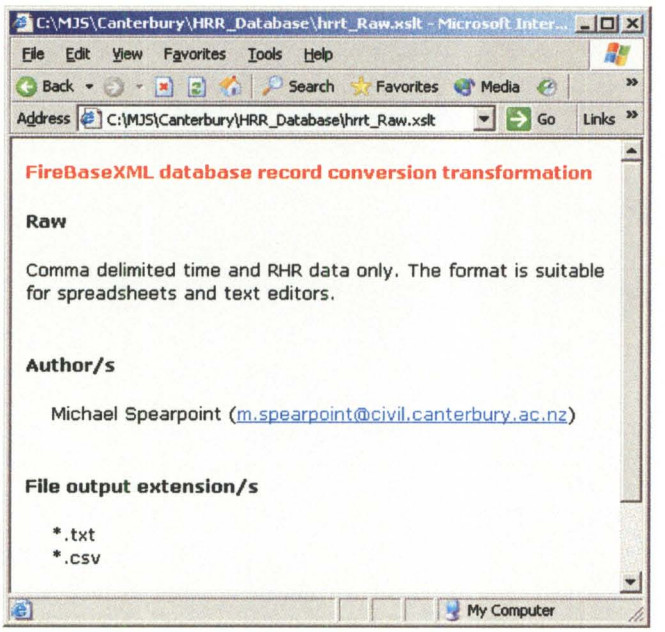


Figure 6-16. A FireBaseXML conversion document viewed as a webpage generated by the viewConversion.xslt transformation.

As discussed in Section 6.2.6, a webpage view of the FireBaseXML schema using XSL transformations can be generated. In a similar manner, the same generic set of templates and a set of templates specific to the FireBaseXML_Conversion.xsd schema to generate a webpage view of the FireBaseXML Conversion schema are used.

6.3.3 FireBaseXML Conversion documents

FireBaseXML conversion transformation documents are XML documents containing a mix of standard XSL script and FireBaseXML Conversion schema elements. The XSL portions perform the transformation of a FireBaseXML record while the FireBaseXML Conversion schema elements describe the conversion. Each of the conversions are detailed in the following sub-sections and the FireBaseXML record shown in Figure 6-17 is used to illustrate the results of each conversion.

```
1:  <item type="discrete" id="EASYF21">
2:      <attribute Armchair="Upholstered"/>
3:      <reference id="NBSIR-82-2604" type="Report" organisation="NIST/NBS">
4:          <title>Upholstered Furniture Heat Release Rates Measured with a
                    Furniture Calorimeter</title>
5:          <document>NBSIR 82-2604</document>
6:          <authors>Babrauskas V, Lawson J R, Walton W D et al</authors>
7:      </reference>
8:      <description>
9:          <summary>Easy chair, F21</summary>
10:             <details>Wood frame easy chair, polyurethane padding, polyolefin
                    fabric.</details>
11:     </description>
12:     <heat_of_combustion type="available_energy" units="J/kg">18100
            </heat_of_combustion>
13:     <initial_mass type="mass" units="kg">28.20</initial_mass>
14:     <data type="Rhr">
19:         <time type="time" units="s">0,155,187,212,250,260,270,280,300,360,
                420,512,675,840,1200</time>
20:         <rhr type="heat_flow_rate" units="kW">0,80,200,365,1860,1940,1930,
                1945,1500,500,250,200,250,160,115</rhr>
21:     </data>
22: </item>
```

Figure 6-17. A typical FireBaseXML record used to illustrate conversion results.

6.3.3.1 Raw

The hrrt_Raw.xslt conversion simply takes the rate of heat release data from a FireBaseXML record and generates a CSV format file which is suitable for

spreadsheets and text editors. The conversion removes any other information associated with the original FireBaseXML record and this would make the output file of very limited use for auditing purposes.

Figure 6-18 shows that lines 5 to 21 conform to the FireBaseXML Conversion schema with a namespace declaration given in line 5 while the remainder of the document consists of XSL script. The <data> and decompose templates in the XSL transformation are very similar to those same templates described in Section 6.2.3 and so are not discussed further here.

```
1:  <?xml version="1.0" encoding="UTF-8"?>
2:  <?xml-stylesheet type="text/xsl" href="hrrt_viewConversion.xslt" ?>
3:  <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
4:      <xsl:output indent='no' method='text'/>

5:      <hrrt:Transformation
            xmlns:hrrt="http://www.civil.canterbury.ac.nz/spearpoint/HRR_Database/
            FireBaseXML_Conversion.xsd">
6:          <hrrt:name>Raw</hrrt:name>
7:          <hrrt:version>2004.1</hrrt:version>

8:          <hrrt:description language='english'>
9:              <hrrt:summary>
10:                 Comma delimited time and RHR data only.
11:             </hrrt:summary>
12:             <hrrt:details>
13:                 Comma delimited time and RHR data only. The format is suitable for
                    spreadsheets and text editors.
14:             </hrrt:details>
15:         </hrrt:description>

16:         <hrrt:author email="m.spearpoint@civil.canterbury.ac.nz">
            Michael Spearpoint</hrrt:author>
17:         <hrrt:output>
18:             <hrrt:extension>txt</hrrt:extension>
19:             <hrrt:extension>csv</hrrt:extension>
20:         </hrrt:output>
21:     </hrrt:Transformation >

        <!--
*************************************************************************************
-->
        <!-- root element -->
22:     <xsl:template match="/">
23:         <xsl:apply-templates select="item"/>
24:     </xsl:template>
```

```
        <!--
*************************************************************************************
-->
        <!-- <item> element -->
25:     <xsl:template match="item">
26:         <xsl:apply-templates select="data"/>
27:     </xsl:template>

        <!--
*************************************************************************************
-->
        <!-- <data> element -->
28:     <xsl:template match='data'>
29:         <xsl:variable name="csv-time" select="time"/>
30:         <xsl:variable name="csv-rhr" select="rhr"/>

31:         <xsl:if test="string-length($csv-time)>0">
32:             <xsl:call-template name='decompose'>
33:                 <xsl:with-param name="t"><xsl:value-of select="$csv-time"/>
                    </xsl:with-param>
34:                 <xsl:with-param name="r"><xsl:value-of select="$csv-rhr"/>
                    </xsl:with-param>
35:             </xsl:call-template>
36:         </xsl:if>

37:     </xsl:template>

        <!--
************************************************************************************* -->

        <!-- decompose template -->
38:     <xsl:template name='decompose'>
39:         <xsl:param name="t"></xsl:param>
40:         <xsl:param name="r"></xsl:param>


41:         <xsl:choose>

                <!-- Check for a comma in the time parameter string -->
42:             <xsl:when test="contains($t, ',')">

                    <!-- A comma has been found, decompose lists and iterate -->
43:                 <xsl:variable name="csv-time" select="substring-after($t,',')"/>
44:                 <xsl:variable name="time" select="substring-before($t,',')"/>
45:                 <xsl:value-of select="format-number($time, '0.0')"/>
46:                 <xsl:text>,</xsl:text>

47:                 <xsl:variable name="csv-rhr" select="substring-after($r,',')"/>
48:                 <xsl:variable name="rhr" select="substring-before($r,',')"/>
49:                 <xsl:value-of select="format-number($rhr, '0.0')"/>
50:                 <xsl:text>&#xA;</xsl:text>

51:                 <xsl:if test="string-length($csv-time)>0">
52:                     <xsl:call-template name='decompose'>
```

```
53:                         <xsl:with-param name="t"><xsl:value-of select="$csv-time"/>
                            </xsl:with-param>
54:                         <xsl:with-param name="r"><xsl:value-of select="$csv-rhr"/>
                            </xsl:with-param>
55:                     </xsl:call-template>
56:                 </xsl:if>

57:             </xsl:when>

                <!-- No comma found so we must be at the last value in our csv lists -->
58:             <xsl:otherwise>
59:                 <xsl:value-of select="format-number($t, '0.0')"/>
60:                 <xsl:text>,</xsl:text>
61:                 <xsl:value-of select="format-number($r, '0.0')"/>
62:                 <xsl:text>&#xA;</xsl:text>
63:             </xsl:otherwise>

64:         </xsl:choose>

65:     </xsl:template>

        <!--
********************************************************************************
-->

66:     </xsl:stylesheet>
```

Figure 6-18. The hrrt_Raw.xslt conversion.

The output from the hrrt_Raw.xslt conversion using the example FireBaseXML record (Figure 6-17) is:

```
0.0,0.0
155.0,80.0
187.0,200.0
212.0,365.0
250.0,1860.0

        [etc.]
```

6.3.3.2 Rhr

The hrrt_Rhr.xslt conversion is almost the same as the hrrt_Raw.xslt conversion except that the two columns of the CSV data are given headings corresponding to their data types. The output from the hrrt_Rhr.xslt conversion using the example FireBaseXML record (Figure 6-17) is:

```
s,kW
0.0,0.0
155.0,80.0
```

```
187.0,200.0
212.0,365.0
250.0,1860.0

      [etc.]
```

6.3.3.3 FPETool

The hrrt_fpetool.xslt conversion is used to generate an input fire file for the **FPETool** fire simulation package (Deal, 1995). The output from the hrrt_fpetool.xslt conversion using the example FireBaseXML record (Figure 6-17) is:

```
0.000000 0.000000 0.000000
155.000000 80.000000 0.004420
187.000000 200.000000 0.011050
212.000000 365.000000 0.020166
250.000000 1860.000000 0.102762

      [etc.]

1200.000000 115.000000 0.006354
-9 -9 -9
EASYF21.FIR
Easy chair, F21
```

The source FireBaseXML record must have an average heat of combustion defined for this transformation to work correctly since it is required to calculate the mass loss at each time step from the rate of heat release rate. Where no heat of combustion is specified the result of the conversion is likely to cause errors. It is possible to overcome the problem of not having a heat of combustion defined in a FireBaseXML record by allowing the user to intervene. With the **SelectFire** application (see Section 5.12.3), if a record without a heat of combustion is extracted, the user is presented with a dialogue box in which they can specify a value.

6.3.3.4 XML

The hrrt_xml.xslt conversion simply extracts a FireBaseXML record as a stand-alone XML document which can be viewed or processed in any appropriate software. The output from the hrrt_xml.xslt conversion using our example FireBaseXML record (Figure 6-17) is shown in a browser in Figure 6-19.

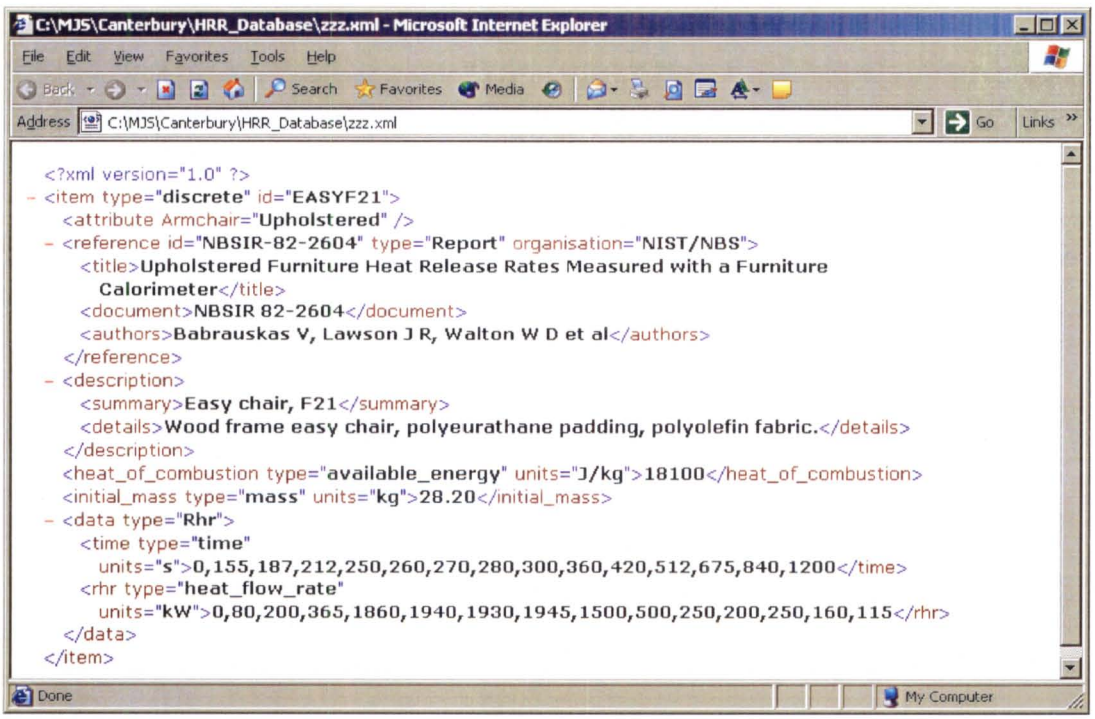


Figure 6-19. The example FireBaseXML record converted using hrrt_xml.xslt and viewed in a web browser.

If the FireBaseXML database document uses idref attributes to create intra-document links between <reference> elements, as described in Section 5.11, then the hrrt_xml.xslt conversion will only extract the <reference> element and its associated idref attribute. However it is possible to expand the <reference> element to include its entire set of associated child elements using the intra-document link. This option to expand the <reference> element is presented to a user when the hrrt_xml.xslt conversion is used by the **SelectFire** application (see Section 5.12.3).

6.3.3.5 Pset

The hrrt_Pset.xslt conversion creates an XML file that conforms to the IFC Property Set Definition (PSD) schema. The purpose of PSDs is given in more detail in Section 7.7. The output from the hrrt_Pset.xslt conversion using the example FireBaseXML record (Figure 6-17) is shown in Figure 6-20.

```
<?xml version="1.0"?>
<?xml-stylesheet type="text/xsl" href="PSD_R30.xsl"?>
<!--Document generated by 'hrrt_Pset.xslt' from a FireBaseXML fragment-->
<!DOCTYPE PropertySetDef SYSTEM "PSD_R30.dtd">
```

```
<PropertySetDef>
    <IfcVersion version="2.x" schema="IfcFireEngineeringDomain"/>
    <Name>Pset_FurnitureHeatRelease</Name>
    <Definition>
            Definition from MJS, University of Canterbury

            HISTORY:

            ISSUES:

        </Definition>
    <Typed>False</Typed>
    <TypedClass/>
    <TypeName/>
    <PropertyDefs>
        <PropertyDef id="1">
            <Name>HeatOfCombustion</Name>
            <PropertyType>
                <TypePropertySingleValue>
                    <DataType type="IfcReal"/>
                    <UnitType type="UserDefined"/>
                </TypePropertySingleValue>
            </PropertyType>
            <ValueDef>
                <DefaultValue value="18100"/>
            </ValueDef>
            <Definition>The average heat of combustion</Definition>
        </PropertyDef>
        <PropertyDef id="2">
            <Name>RateOfHeatRelease</Name>
            <PropertyType>
                <TypePropertyTableValue>
                    <DefiningValue>
                        <DataType type="IfcTimeMeasure"/>
                        <UnitType type="TIMEUNIT"/>
                        <Values>
                            <ValueItem>0.0</ValueItem>
                            <ValueItem>155.0</ValueItem>
                            <ValueItem>187.0</ValueItem>
                            <ValueItem>212.0</ValueItem>
                            <ValueItem>250.0</ValueItem>

                                [etc.]

                        </Values>
                    </DefiningValue>
                    <DefinedValue>
                        <DataType type="IfcEnergyMeasure"/>
                        <UnitType type="ENERGYUNIT"/>
                        <Values>
                            <ValueItem>0.0</ValueItem>
                            <ValueItem>80.0</ValueItem>
                            <ValueItem>200.0</ValueItem>
                            <ValueItem>365.0</ValueItem>
                            <ValueItem>1860.0</ValueItem>

                                [etc.]

                        </Values>
                    </DefinedValue>
```

```
            </TypePropertyTableValue>
        </PropertyType>
        <Definition>Time series rate of heat release</Definition>
    </PropertyDef>
    <PropertyDef id="3">
        <Name>OriginalTestID</Name>
        <PropertyType>
            <TypePropertySingleValue>
                <DataType type="IfcText"/>
            </TypePropertySingleValue>
        </PropertyType>
        <ValueDef>
            <DefaultValue value="EASYF21"/>
        </ValueDef>
        <Definition>The original ID of the test taken from the FireBaseXML
            database</Definition>
    </PropertyDef>
  </PropertyDefs>
</PropertySetDef>
```

Figure 6-20. The example FireBaseXML record converted using hrrt_Pset.xslt.

6.3.3.6 BRANZFIRE

The hrrt_branzfire.xslt conversion is used to directly interface from the **BRANZFIRE** fire simulation application to a FireBaseXML database and is described in more detail in Section 6.4.2.1.

6.4 A BRANZFIRE interface to FireBaseXML

In this section a Visual Basic control that interfaces to a FireBaseXML database is discussed. This generic control and an associated XSL transformation are then used to create an interface specifically written for the **BRANZFIRE** room fire simulator code (Wade, 2002).

6.4.1 Generic interface control

FireBaseXMLIx is a **Visual Basic** control that interfaces to a FireBaseXML database. The control is written using Microsoft **Visual Basic 6.0** and exposes several events and methods that allow a programmer to use the control from within their own program. The interface includes windows to search a database, view records and extract records and these are derived from the **SelectFire** application (Section 5.12) source code. The control forms a basic building block from which specific tools can be developed as described below. It is distributed as an .ocx file and can be

imported into a **Visual Basic** project via the *Components* window that is accessed through the *Project→ Components...* menu item.

6.4.1.1 Events

The control provides a number of events that the programmer can respond to. Several events typical to any control are available such as `DragDrop`, `DragOver`, `GotFocus`, `LostFocus` and these are not detailed further here.

An additional event unique to this control called `DataReady` is also provided. This event is raised after a user has successfully selected a record from a database. Typical usage of this event is described below.

6.4.1.2 Methods

In addition to methods normally associated with a control, the **FireBaseXMLIx** control includes several specific methods. Two methods are provided so that the programmer can specify the FireBaseXML database URL and also the default transformation URL. These methods are called `setDatabaseURL` and `setTransformationURL` respectively. The following code example specifies the database and transformation to be used when a form containing the interface control is loaded.

```
Private Sub Form_Load()
    FireBaseXMLIxCtrl.setDatabaseURL
           ("http://www.civil.canterbury.ac.nz/spearpoint/
           HRR_Database/HRR_Database.xml")
    FireBaseXMLIxCtrl.setTransformationURL
           ("http://www.civil.canterbury.ac.nz/spearpoint/
           HRR_Database/hrrt_xml.xslt")
End Sub
```

The method called `getXMLDocument` allows the programmer to read the XML document that was the result of the user selecting a record from a FireBaseXML database. Typically this method would be used on responding to a `DataReady` event. For example, the following code opens a message box that contains the text portion of the XML document that resulted from a user selection from a database

```
Private Sub FireBaseXMLIxCtrl_DataReady()
```

```
    MsgBox (FireBaseXMLIxCtrl.getXMLDocument.Text)
End Sub
```

6.4.2 BRANZFIRE interface control

The **FireBaseXMLIxBF** control is an extension to the **FireBaseXMLIx** generic control specifically written to allow access to a FireBaseXML database from **BRANZFIRE**. The extended control includes specific methods that provide the rate of heat release data in a form that can be directly integrated with **BRANZFIRE**. Only minimal changes are required to the core **BRANZFIRE** code to implement the **FireBaseXMLIxBF** control.

BRANZFIRE contains its own internal database of fire objects. These objects include fields that specify the rate of heat release with time, a description of the object and other appropriate data. The **BRANZFIRE** database categorises the objects into a number of types. These types are similar to the 'groups' specified in the FireBaseXML schema.

6.4.2.1 The **BRANZFIRE** *transformation*

In order to extract a record from a FireBaseXML database in a form that the **FireBaseXMLIxBF** control can then process, a specific XSL transformation has been developed (called hrrt_branzfire.xslt, see Section 6.3.3.6) using the techniques described in Section 6.3. This transformation takes a FireBaseXML conforming XML document fragment (i.e. the extracted FireBaseXML record) and converts it into an XML document that **FireBaseXMLIxBF** can interpret. The advantage of this approach is that if the FireBaseXML schema is changed then a new version of the **BRANZFIRE** XSL document can be published on a server. This means that the **BRANZFIRE** code does not need to be recompiled if the FireBaseXML database structure changes. An example of the XML output of the **BRANZFIRE** XSL transformation is given in Appendix D.2.

6.4.2.2 Data structure

In order to transfer data from the FireBaseXMLIxBF control into **BRANZFIRE**, a new data structure has been defined in FireBaseXMLIxBF. This data structure holds the description of the test, the heat of combustion, the object type and the rate of heat

release data in a suitable format. Access to this data structure is made with additional methods built into **FireBaseXMLIxBF** that are not included in **FireBaseXMLIx**. These methods are `getDescription()`, `getTabdata()`, `getHeatOf-Combustion()` and `getObjectType()` and are described in Appendix D.5.

6.4.2.3 Implementation

The **FireBaseXMLIxBF** control has been added to **BRANZFIRE**'s `frmFireData` form. This allows the user to access a FireBaseXML database when they are creating or editing a fire object record in the **BRANZFIRE** database. When the from is loaded, the FireBaseXML database URL and **BRANZFIRE** transformation URL are specified as shown in the code in Appendix D.3. When the `DataReady` event is raised from **FireBaseXMLIxBF** then the `frmFireData` form responds by filling in the appropriate form fields as shown by the code in Appendix D.4.

6.4.3 Using the interface in BRANZFIRE

Importing a fire from FireBaseXML will only populate those text boxes in the Fire *Object Database* window for which data is available. The user will need to manually add any other fields before saving the record to the **BRANZFIRE** database. The record will not be saved if any of the fields are left blank. The process of transforming a FireBaseXML record into the **BRANZFIRE** *Fire Object Database* is as follows:

1. The user wants to add or edit a **BRANZFIRE** fire object by opening the *Fire Object Database* window (defined by the `frmFireData` form).
2. As the form is loaded, the FireBaseXML database URL and **BRANZFIRE** transformation URL are set using the `setDatabaseURL` and `setTransfsormationURL` methods respectively
3. The user clicks on the **FireBaseXMLIxBF** control button on the *Fire Object Database* window.
4. User navigates the FireBaseXML database, selects the desired record and extracts it from the database (Figure 6-21).
5. The selected record is transformed using the specified XSL document into a form that **FireBaseXMLIxBF** can process.

6. **FireBaseXMLIxBF** fills the `FireBaseXMLIxCtrl` data structure with the relevant information.
7. The FireBaseXMLIxBF control raises a `DataReady` event that is responded to by the `frmFireData` form.
8. The `frmFireData` form transfers the contents of the `FireBaseXMLIxCtrl` data structure into the appropriate *Fire Object Database* window text fields and blanks out those that are not filled.

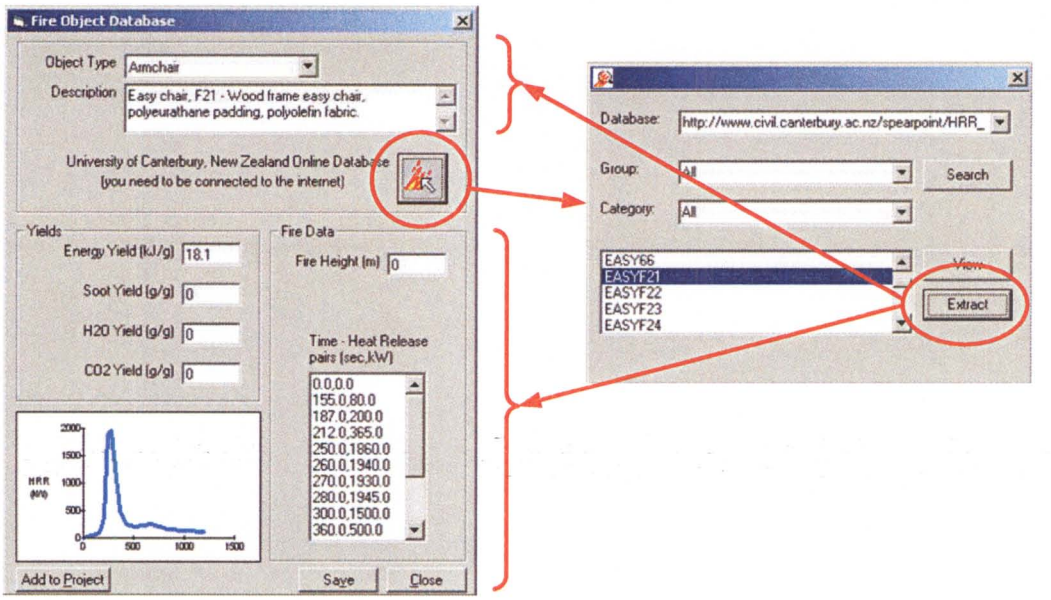


Figure 6-21. User interface between **BRANZFIRE**'s *Fire Object Database* window and a FireBaseXML database.

Chapter 7:

PROPERTIES FOR FIRE ENGINEERING DESIGN IN NEW ZEALAND AND THE IFC BUILDING PRODUCT MODEL

Spearpoint M. Properties for fire engineering design in New Zealand and the IFC building product model. CIB W78 20th International Conference on Information Technology in Construction, Auckland, New Zealand, 2003.

Paper Referees:

Three anonymous referees.

Abstract

This paper examines the information needs of fire engineers and relates those needs to the IFC Building Product Model. It identifies what is already provided in the IFC 2x Model and how in particular it corresponds with the New Zealand Approved Document Acceptable Solution C/AS1. The paper then demonstrates how Property Set Definitions can be used to extend the scope of the IFC Model for use by fire engineers.

7.1 Fire engineering design

Fire engineering is only one of the domains in the AEC/FM industry that can benefit from electronic descriptions of buildings. It is a specialised discipline that requires its own domain specific elements as well as elements common to many of the other construction related domains.

Fire engineers are involved in many aspects of a building's construction, fit-out and potential renovation. The New Zealand Building Code is a performance-based code that includes fire safety requirements with the objectives of providing means of escape to occupants, preventing the spread of fire to neighbouring property, providing protection to fire service personnel during fire-fighting and limiting the effects of fire on the environment. These objectives are met through the consideration of issues such as exit route design, fire and smoke spread mechanisms and structural stability and can be achieved by following published acceptable design methods or by considering alternative solutions using fire-specific design.

In order to carry out alternative designs, fire engineers are likely to conduct computer simulations particularly where the building is complex and there may be several fire scenarios that need to be considered. Much of the initial simulation effort is spent simply obtaining and transferring the basic building description into the specific fire/evacuation application. Very few of the currently available applications can import building information in an electronic form and even those that do are limited by the information available such as provided in a DXF file.

7.2 Common elements

There are many aspects of a building that are common to the fire engineering domain and other domains such as architecture, structural engineering and building services. Fire engineers need to have the basic geometry and topology of a building. This includes information such the size and shape of rooms, openings and hidden voids that could be a means of fire or smoke spread, the exits from a space and where those exits lead.

Fire engineers are involved in the specification and design of specific fire safety systems such as alarm, suppression and smoke management systems. These will require details of system components (such as smoke detectors, sprinklers, fans) plus electrical wiring layouts, plumbing and pipe work, ducting networks etc.

Fire engineers need to determine the fires that could likely occur through an assessment of the fuels in the building. This requires the fire engineers to determine the fire properties of lining materials, the contents of the spaces in terms of total fuel load, the arrangement of fuel packages and the relative flammability of those packages. Fuel packages might include furniture and fittings, stored items plus wall, floor and ceiling coverings.

Information regarding the site of the building may also be necessary. For example weather may be a factor and temperatures, wind velocities, humidity may all be required in order to specify the performance of the fire safety systems.

Finally fire engineers need to obtain details of the occupancy characteristics of the building. This may include information such as the primary use of the spaces, numbers of people, times when the building will be occupied and by whom, the physical and mental state of the occupants.

7.3 Property categories

Within the context of fire safety, the properties of building elements can be thought of as belonging to three general categories.

- Category 1: the fundamental thermo-physical properties of a building element. These properties might include the thermal conductivity, specific heat capacity and so on.
- Category 2: fire specific properties that may have been obtained by measurement or some other means. Here we might include such things as the heat release rate (i.e. the energy release over time) and the properties of the smoke generated by a burning item.
- Category 3: properties that have been obtained for regulatory or standardisation purposes. These properties are generally obtained from some form of standard test method and are derived properties that have a specific regulatory meaning.

7.4 IFC Building Product Model

The IFC Building Product Model is a general product model that provides an object-oriented description of many aspects of a building and related services enabling interoperability between different vendors of AEC/FM software. It aims to support the exchange of information throughout the design, construction and operation stages of a project. Development of the IFC Model began around 1996 and has continued through several versions up to the present IFC 2x release as specified by Liebich et al. (2000).

The IFC Model has the potential to allow fire engineers to automate compliance checking with published acceptable methods by the use of appropriately designed software applications. The IFC Model also can be used to allow fire engineers to extract relevant building-related information in order to conduct alternative designs using fire modelling applications. In each case, the form and level of detail available in the IFC Model will dictate the scope of either method. Mapping a general product model, such as the IFC Model, to a highly domain specific application can present limitations as demonstrated by Karola et al. (2002).

Many of the Category 1 fundamental properties that are useful for fire engineers are already available in IFC 2x. These include *IfcEnergyMeasure*, *IfcHeatFlux-DensityMeasure*, *IfcSpecificHeatCapacityMeasure*, *IfcThermalConductivityMeasure*

etc. There is scope to extend these fundamental properties so as to include additional ones that would be useful for fire engineers. This might include properties such as heat of combustion, apparent ignition temperature or the ability to have properties that are a function of some other parameter, for example, the thermal conductivity as a function of temperature. However many of these might be better seen as Category 2 properties as they might not be considered fundamental properties.

IFC 2x already recognises the fact that buildings contain elements with properties specifically related to fire engineering and these are shown in Table 7-1. Examining these properties closely we find that the majority of them are essentially Category 3 regulatory properties.

7.5 Property Set Definitions

As a general product model, it is not the intention of the IFC Model to provide a class for every type of object that could be encountered. Instead, object types are generalised at a relatively high level and these object types terminate at 'IFC Leaf Nodes' (Figure 7-1), from Liebich and Wix (2000).

In order to provide detailed properties for individual disciplines, domain specific models could be developed to supplement the general product model. The use of general product models or domain specific models have certain advantages and disadvantages as pointed out by Ito (1995).

Property set definition and associated IFC class	Fire specific property name	Data type	Definition
Pset_DoorCommon (*IfcDoor*)	*FireRating*	*IfcString*	Fire rating of complete door assembly. Given according to the national fire safety classification.
Pset_Insulation (*IfcDiscreteElement*)	*FlamabilityRating*	*IfcString*	Insulation flammability rating.
Pset_CoveringCommon (*IfcCovering*)	*FireRating*	*IfcString*	Rating indicating the time duration before fire would penetrate this ceiling
Pset_RoofCommon (*IfcRoof*)	*FireRating*	*IfcString*	Time duration for fire resistance the roof assembly is rated.
Pset_SlabCommon (*IfcSlab*)	*FireRating*	*IfcString*	Fire rating of slab.
Pset_SpaceCommon (*IfcSpace*)	*MainFireUse*	*IfcString*	Main fire use for the space which is assigned from the Fire Use Classification.
	AncillaryFireUse	*IfcString*	Ancillary fire use for the space which is assigned from the Fire Use Classification.
	FireRiskFactor	*IfcInteger*	Fire Risk factor assigned to the space
	SprinklerProtection	*IfcBoolean*	Indication whether the space is sprinkler protected (true) or not (false).
Pset_StairCommon (*IfcStair*)	*FireRating*	*IfcString*	Fire survival rating = length of time the stair enclosure/ assembly will survive in case of fire
	ExitStair	*IfcBoolean*	Is this stair counted as an exit stair in case of fire?
Pset_WallCommon (*IfcWall*)	*FireRating*	*IfcString*	Fire rating of wall assembly.
Pset_WindowCommon (*IfcWindow*)	*FireRating*	*IfcString*	Fire rating of complete window assembly. Given according to the national fire safety classification.
Pset_FireDamper, *Pset_FireSmokeDamper* (*IfcDamper*)	*FireResistance-Rating*	*IfcReal*	Measure of the fire resistance rating in hours (e.g., 1.5 hours, 2 hours, etc.).
	FusibleLink-Temperature	*IfcThermo-dynamic-Temperature-Measure*	The temperature that the fusible link melts.
	ControlType	*IfcString*	The type of control used to operate the damper (e.g., Open/Closed Indicator, Resetable Temperature Sensor, Temperature Override, etc.)
	Plus other associated properties...		
Pset_SmokeDamper (*IfcDamper*)	*ControlType*	*IfcString*	The type of control used to operate the damper.
	Plus other associated properties...		

Note: not all damper properties shown.

Table 7-1. IFC 2x fire-related properties.

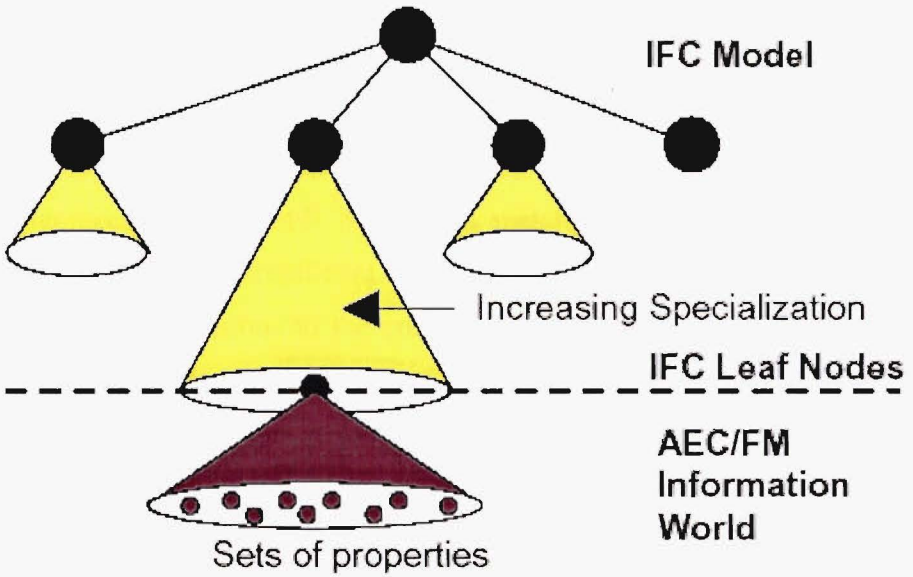


Figure 7-1. Limits of the IFC Model.

The IFC Model allows additional property sets to be attached to leaf nodes so that more detailed sets of properties can be accommodated in the model. These property sets must conform to the Property Set Definition (PSD) sub-schema of the IFC Model. Specialists within segments of the construction industry can carry out the task of defining property sets for IFC Model objects since such specialists will know what properties are relevant to a specific object.

7.6 New Zealand approved documents

The Approved Document Acceptable Solution C/AS1 published by the BIA (2001) is one method in which a fire engineer can demonstrate that a building complies with the performance criteria of the New Zealand Building Code. C/AS1 is a set of design methods that address the fire safety requirements of buildings and includes a number of defined terms that have a specific regulatory meaning (shown in italics here and in C/AS1). C/AS1 is used by fire engineers for many simple, low-rise buildings in New Zealand.

We have already noted from Table 7-1 that the majority of fire specific properties in IFC 2x belong to our Category 3 regulatory properties. In the case of providing an automated compliance checking software tool, we investigate here how closely the fire-related properties in the IFC Model integrate with the regulatory requirements of C/AS1. In particular we are interested in finding where the IFC Model and C/AS1 have common components or where they differ and if so by how much. If there are differences we may still be able to match the IFC Model with C/AS1 if those differences are not significant. However if there are major differences, we may have to consider extensions to the core IFC Model by using the Property Set Definition mechanism.

7.6.1 Fire resistance

In most buildings certain elements are provided with a fire resistance in order to prevent the spread of fire and smoke or avoid structural collapse during a significant fire. In many cases this fire resistance is obtained from a standard furnace test such as ISO-834 (International Organization for Standardization, 1999) and is specified as a failure time in minutes. C/AS1 specifies fire resistance ratings ("*FRR*") using three numbers indicating values for "*stability*", "*integrity*" and "*insulation*". In some cases all three numbers will be the same, in others the numbers will differ and some may have a value of zero.

The *FireRating* property in IFC 2x allows the specification of the "*FRR*" to be provided for certain building elements. The *FireRating* property is used by the *IfcWall*, *IfcDoor*, *IfcRoof*, *IfcStair*, *IfcWindow*, *IfcSlab* and *IfcCovering* entities through several property set definitions as shown in Table 7-1. The use of the *IfcString* type for this property means that the "*FRR*" designation in the form of '*x*/*x*/*x*', where '*x*' are times in minutes for "*stability*", "*integrity*" and "*insulation*", can be accommodated.

7.6.2 Space utilisation

In order to assess the number and mobility of occupants, the activities undertaken and the nature of building materials and contents, C/AS1 categorises building spaces into "*purpose groups*" and "*fire hazard categories*" ("*FHC*"). C/AS1 contains around 16

specific "*purpose groups*" subdivided into four major activity sets that identify the broad use of the space. The four major activity sets are crowd, sleeping, working business or storage and intermittent. Examples of "*purpose groups*" for each activity set are "Crowd Large (CL)" for a cinema, "Sleeping Accommodation (SA)" for a hotel, "Working Low (WL)" for a factory containing materials that burn slowly and "Intermittent Activity (IA)" for a car park. Each "*purpose group*" also has an associated "*FHC*" designation that identifies the fuel characteristics in the space.

IFC 2x does not specifically contain properties for "*purpose groups*" and "*fire hazard categories*". However the *Pset_SpaceProgramOccupied* property set, used by the *IfcSpaceProgram* entity, contains a *BldgCodeOccupancyType* as an *IfcString* that essentially fulfils the role of identifying the purpose group. For "*FHC*", the *Pset_SpaceCommon* property set associated with the *IfcSpace* entity has two properties named *MainFireUse* and *AncillaryFireUse*, both of type *IfcString* type, in which either could be used to hold the "*fire hazard category*". Alternatively a new property set specifically for C/AS1 could be created outside the core IFC 2x Model in which "*purpose groups*" and "*fire hazard category*" are explicitly defined.

7.6.3 Exit routes

For the design of escape routes C/AS1 has a number of specific definitions. These include "*escape route*", "*dead end*", "*open path*", "*protected path*", "*safe path*", "*exitway*" and "*final exit*". IFC 2x only has one property relevant to escape route design and this is the *ExitStair* property in *Pset_StairCommon* used by the IfcStair class. Although this property name is similar to the C/AS1 *exitway*, the specific regulatory definition in C/AS1 is "all parts of an "*escape route*" protected by "*fire*" or "*smoke separations*", or by distance when exposed to open air, and terminating at a "*final exit*". This means that there could be ambiguity where a stair forms part of the "*escape route*" but is not the whole "*exitway*". It is clear that significant additions would need to be made to the IFC Model in order to incorporate the C/AS1 definitions for exit routes and this could only be achieved by the introduction of new property sets for spaces and stairs.

7.6.4 Material flammability

In addition to providing physical separations and the installation of fire safety systems, the control of internal fire and smoke spread can include an assessment of the burning characteristics of materials within a space. Such materials include surface finishes to walls, floor coverings, suspended fabrics and acoustic or thermal insulation materials.

C/AS1 provides a number of requirements for the burning characteristics of nearly all such materials through its "spread of flame index" (SFI), "smoke developed index" (SDI) and "flammability index" (FI). These indices are obtained from tests to AS/NZS 1530 (Standards New Zealand, 1999) parts 1-3 where the results are expressed as an integer value. Materials do not necessarily need to have all three indices specified depending on the requirements given by C/AS1. Floor coverings and membrane structures require separate tests in order to assess their burning characteristics and the outcomes of these tests are reportedly differently to the AS/NZS 1530 indices.

IFC 2x only has one instance where material flammability is explicitly referenced and this is in the *Pset_Insulation* property set in which the *FlamabilityRating* property is provided as an *IfcString*. The *Pset_Insulation* property set is used by the *IfcDiscreteElement* class which defines elements in a building services system such as insulation or attaching elements

C/AS1 requires that pipe insulation and acoustic treatments only be assessed in terms of their SDI and SFI and so the *FlamabilityRating* property has little use in conjunction with C/AS1. It is clear that all relevant elements in the IFC Model need to have new property sets in order to include the flammability requirements of C/AS1.

7.6.5 Dampers

Dampers are used to automatically close off an airway between fire-separated parts of a building. C/AS1 defines a "*fire damper*" and requires it has a specified "*FRR*". IFC 2x includes properties for three types of fire-related damper all used by the *IfcDamper* class. The *Pset_FireDamper*, *Pset_FireSmokeDamper* both contain a property called *FireResistanceRating* however, unlike the *FireRating* property for walls etc., this

property is of type *IfcReal* which means it is not able to hold the '*x*/*x*/*x*' style "*FRR*" classification required in C/AS1.

7.7 Heat release Property Set Definition

Where a fire engineer chooses to provide an alternative solution to the published Acceptable Solutions they will often need to examine the fire and smoke conditions in a building. They will frequently have to devise one or more credible worst-case scenarios that will include a 'design fire curve'. Specifying a design fire curve requires certain properties of materials and items that may burn. Babrauskas and Peacock (1992) suggest that the most important of those material properties is the Heat Release Rate (HRR). The HRR can be obtained from basic material properties for simple fuels such as hydrocarbon liquids but more typically from experiments for fuels found in buildings such as furniture and linings.

Databases of experimental HRR are available and in particular the XML-based database called FireBaseXML developed in Chapter 5 is relevant here. The database is already being used from within the **BRANZFIRE** fire modelling software developed by Wade (1999) but is also accessible through web-browser tools or alternative software.

IFC *R3.0* Property Set Definition Reference

PropertySet Definition:

PropertySet Name	*Pset_FurnitureHeatRelease*
Typed	False
TypedClass	
TypeName	
Definition	Definition from MJS, University of Canterbury

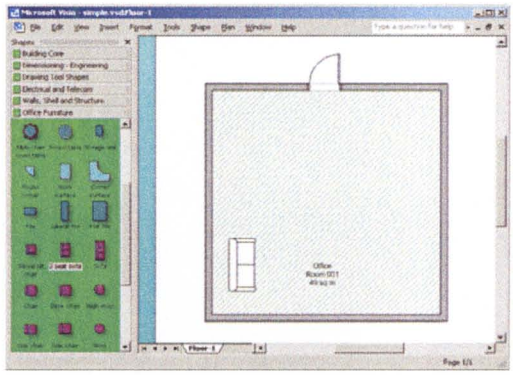
Property Definitions:

Name	Property Type	Data Type	Definition
HeatOfCombustion	*IfcPropertySingleValue*	*IfcReal* / UserDefined	The average heat of combustion
RateOfHeatRelease	*IfcPropertyTableValue*	**Defining Value:** *IfcTimeMeasure* / TIMEUNIT **Defined Value:** *IfcEnergyMeasure* / ENERGYUNIT	Time series rate of heat release
OriginalTestID	*IfcPropertySingleValue*	*IfcText*	The original ID of the test taken from the FireBaseXML database

Note: Defining and Defined Values not shown.

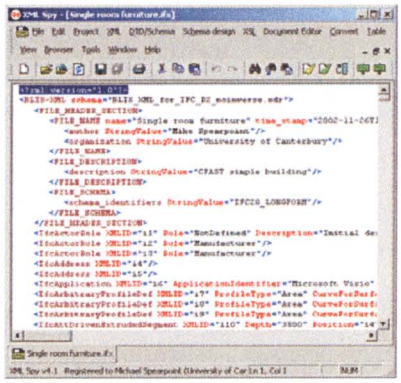
Figure 7-2. A *Pset_FurnitureHeatRelease* Property Set Definition.

Data from a FireBaseXML database can be transformed to an appropriate IFC PSD through a specific XSL transformation and Figure 7-2 shows the content of a *Pset_FurnitureHeatRelease* PSD formatted using the XSL transformation developed by Adachi (2002). The PSD contains the necessary heat release rate data as an *IfcPropertyTableValue*, the average heat of combustion as an *IfcPropertySingleValue* (the units are J/kg and so are specified as 'UserDefined' since these units do not currently exist in the IFC 2x Model) plus a reference to the original item ID in the FireBaseXML database (see Section 7.10).

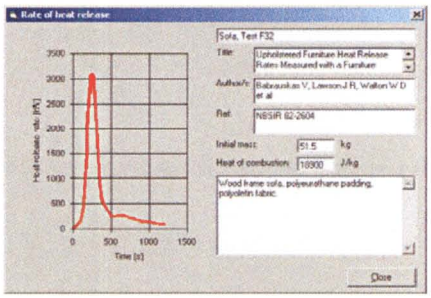


MS Visio 2002 - Simple building with an item of furniture included.

IFC Model Exchange export



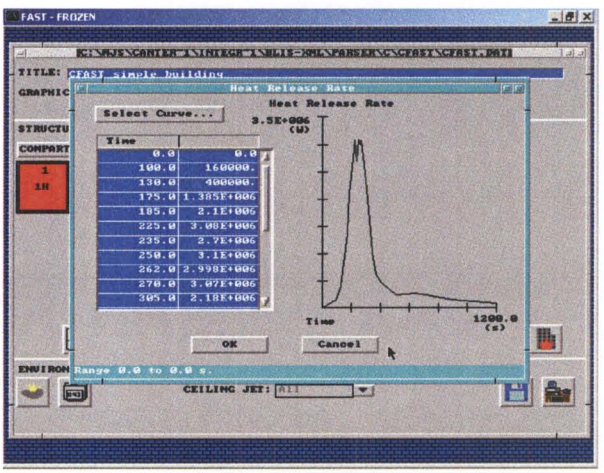
BLIS-XML document.



Fire data for a sofa selected from FireBaseXML database.

Conversion to PSD via XSL

BLIS-XML document parser for FAST



FAST - Building description and fire properties transferred from BLIS-XML document.

Figure 7-3. The IFC Model exchange process into the **FAST** fire modelling application.

The *Pset_FurnitureHeatRelease* PSD can be added to an IFC Model instance and assigned to one or more items of furniture (and this is described in more detail in Section 7.11). Once added to the IFC Model instance, the heat release rate for items can be used when transferring the building description into a fire modelling application.

An example is shown here where a simple building has been created using the BLIS-XML specification developed by BLIS Project Companies (2002). Although the BLIS-XML specification uses IFC Release 2.0 and not IFC 2x the principles are the same. A simple square one-room building was created in Microsoft **Visio 2002** in which a single item of furniture was included (Figure 7-3). This building was exported as a BLIS-XML document using the IFC Model Exchange functionality provided by **Visio**. An appropriate HRR curve was extracted from a FireBaseXML database, converted to a *Pset_FurnitureHeatRelease* PSD and added to the BLIS-XML document. This document is then translated into an input file for the **FAST** (**Fire And Smoke Transport**) fire modelling application developed by Bukowski et al. (1989) in which a furniture item and its associated heat release rate have been included. The dimensions of the room, the exit from the room and the wall material properties are also transferred from the BLIS-XML document to the **FAST** model. The fire modelling application is now ready for the fire engineer to continue with their fire modelling and design.

7.8 Conclusion

The current IFC 2x model provides much general information useful to the fire engineer. The fire specific properties that currently exist in the IFC Model have limited scope when compared to the needs of fire engineers and the ability of the properties to describe the New Zealand C/AS1 requirements. Property Set Definitions can be used to extend the IFC Model so as to include additional fire-related properties. We have shown how we are able to link a HRR database to an IFC Model document using a PSD and transfer the document to a fire modelling package. Significant further work is required to enhance the fire specific properties of the IFC Model either through additional PSDs or eventually by updating the core IFC Model.

Additional comments:

The following sections were not included in the original paper in order to make the document fit the publication length limitations.

7.9 Property category classification

Section 7.3 introduced three broad categories of property that are useful in the context of fire safety and engineering. This classification is particularly relevant to the practice of fire engineering within the building regulatory environment. At the 'deemed-to-satisfy' or 'prescriptive' level of regulatory compliance, only Category 3 properties would likely be needed to evaluate regulatory compliance. For performance-based design involving fire simulations, Category 1 and/or Category 2 properties are desired.

For a fire engineer wishing to conduct simulations with the use of one or more software tools then the provision of Category 1 and/or Category 2 properties would allow the automatic transfer of those properties to the simulation software. This would require that such properties are available in the product model in a form appropriate to fire engineering and that values for these properties were available and present in a specific instance of the product model. Providing such properties would make the task of conducting simulations more efficient and would therefore meet the one of the major objectives of the BIM approach. Category 3 properties are suited to tools in which a building is assessed against 'deemed-to-satisfy' requirements. Regulatory assessment tools could automatically determine whether a particular design meets the requirements by matching Category 3 properties against the prescriptive rules.

Ideally all three Category levels should be included in the product model. Some property data would need to be provided by a human at some stage of the project whilst other properties might be determined automatically by some software tool using the properties already specified. For example, a property such as the SFI (spread of flame index) as discussed in Section 7.6.4 might be determined through a simulation analysis of the fundamental flame spread properties of an item already included in the project. The SFI would then be appended to the project and then another software tool might be used to evaluate regulatory compliance.

It is difficult to give a comprehensive list of every property required by fire engineers in each of the three categories without knowing what analysis a fire engineer wishes to conduct, what software tools they have available and what regulatory environment they are working in. Some examples of Category 1 property requirements are given in Appendix A.1.5 where material properties for boundary elements are given for the **FPEtool** and **CFAST** fire simulation tools are noted. Category 2 properties include such information as rate of heat release, product yields and smoke characteristics of fuels. The rate of heat release property is discussed throughout this thesis and other properties of this type can be determined, for example, from those described in Section 5.8. A range of Category 3 regulatory properties are already provided in IFC 2x2 such as *FireRating*, *SurfaceSpreadOfFlame* discussed in Chapter 9. A complete review of C/AS1 would need to be undertaken to determine whether additional Category 3 properties are needed within the New Zealand regulatory framework. However, Section 7.12 already shows where IFC 2x2 does meet the escape route definitions employed by C/AS1.

7.10 FurnitureHeatRelease property set definition

The FurnitureHeatRelease property set definition contains the information that is necessary in order to provide sufficient details to a fire simulation application. Essentially this information is the time-series heat release; however additional information is included in the property set definition for completeness. Figure 7-2 has already shown the *Pset_FurnitureHeatRelease* property set definition that conforms to the PSD_R30.dtd schema formatted using the PSD_R30.xsl transformation both from Adachi (2002).

In order to create the *Pset_FurnitureHeatRelease* property set definition a set of rate of heat release data needs to be obtained. Clearly, one source of such data is a FireBaseXML database but there may be others. A FireBaseXML record can be converted to a FurnitureHeatRelease property set definition using the hrrt_Pset.xslt transformation (see Section 6.3.3.5) with the following mappings provided:

FireBaseXML record	FurnitureHeatRelease property definition
<item:id> attribute	OriginalTestID
<heat_of_combustion> element	HeatOfCombustion
<time> element	RateOfHeatRelease – Defining Value
<rhr> element	RateOfHeatRelease – Defined Value

Figure 7-4. FireBaseXML record to ***Pset_FurnitureHeatRelease*** property definition mappings.

The **SelectFire** application (see Section 5.12) can be used to select a FireBaseXML item and perform the necessary conversion through the hrrt_Pset.xslt transformation.

7.11 Linking the FurnitureHeatRelease property set definition with a BLIS-XML document

Once a FurnitureHeatRelease property set definition has been created it can be appended to an existing BLIS-XML document. This section describes an approach to achieve this.

The xml-stylesheet reference (see Section 6.2) is changed from the default PSD_R30.xsl which is used to format and view the property set definition to Ifc2Pset.xslt which is used to convert the document into a BLIS-XML property set and associated properties. Microsoft **Visio 2002** uses the IFC 2.0 implementation of BLIS-XML but the Ifc2xPset.xslt transformation can be used to create a similar IFC 2x property set. Figure 7-5 shows the transformation process diagram for Ifc2Pset.xslt (version 2002.1) which is sufficient to convert a FurnitureHeatRelease PSD to an equivalent set of BLIS-XML compatible XML elements. The transformation of an IFC 2x property set using Ifc2xPset.xslt follows the same basic process as shown for Ifc2Pset.xslt but with minor differences to the resultant XML elements.

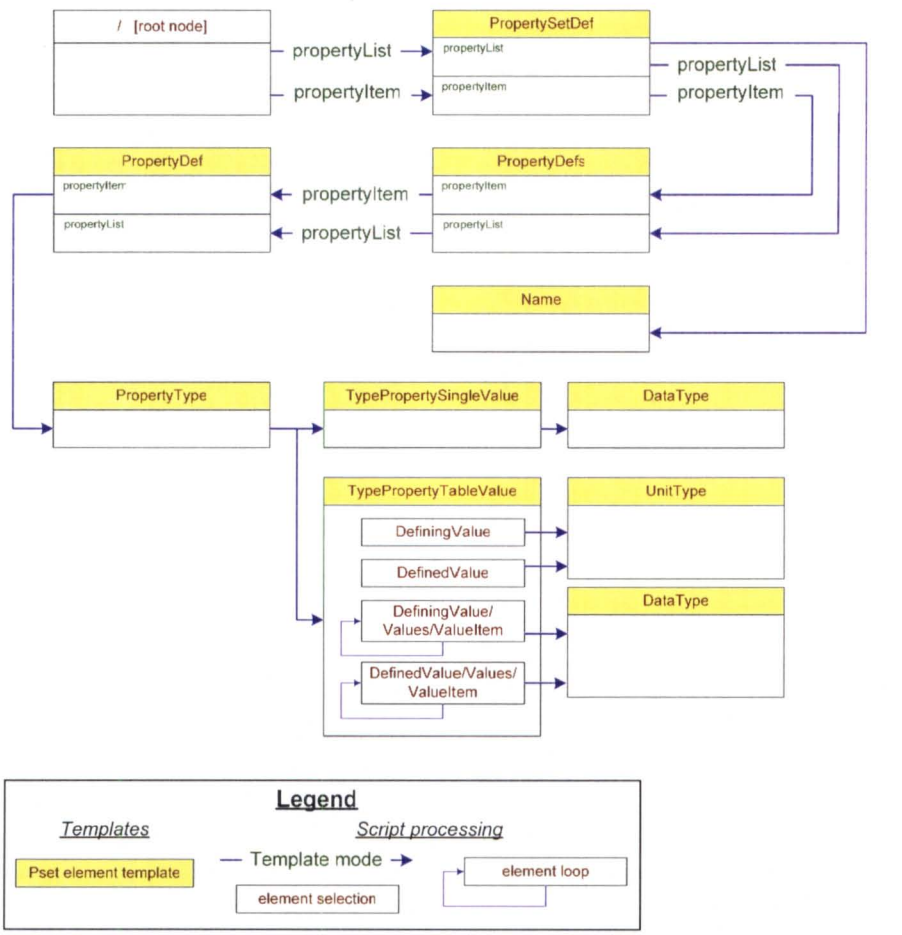


Figure 7-5. Transformation process diagram for Ifc2Pset.xslt and Ifc2xPset.xslt.

The resultant XML fragment is appended to the required BLIS-XML document. The XMLID attributes in the fragment must not have the same references as any of the XMLID attributes in the BLIS-XML document otherwise parsing errors are likely to occur.

The appended property set is linked to the furniture item by using a <IfcRelAssignsProperties> element. The element must be given a unique XMLID attribute, the RelatingPropertyDefinition attribute points to the XMLID of the <IfcPropertySet> element created in the XML fragment and the RelatedObjects attribute points to the XMLID of the corresponding <IfcFurniture> element. For example:

```
<IfcFurniture XMLID="i261" GlobalId="L803&amp;amp;-_+;^x/:wjy/r=o"
        OwnerHistory="i319" Label="2-arm seat module" DocumentReferences=""
        LocalPlacement="i293" Representations="i333" PredefinedType="Chair"
        AssignedTo="" FurnitureModel="i262"/>

<IfcRelAssignsProperties XMLID="i8999" GlobalId="mjs1" OwnerHistory="i319"
        RelatedIsDependent="0" RelatingIsDependent="1"
        RelatingPropertyDefinition="i9000" RelatedObjects="i261"
        DomainView="Generic"/>

<IfcPropertySet XMLID="i9000" GlobalId="mjs2"
        Name="Pset_FurnitureHeatRelease"
        HasProperties="i9001 i9002 i9003 "/>

<IfcSimpleProperty XMLID="i9001" Name="HeatOfCombustion">
    <ValueComponent>
        <IfcMeasureValue>
            <IfcReal RealValue="18900"/>
        </IfcMeasureValue>
    </ValueComponent>
</IfcSimpleProperty>

<TableProperty XMLID="i9002" Name="RateOfHeatRelease"
        DefiningUnit="TIMEUNIT" DefinedUnit="ENERGYUNIT">
    <DefiningValues>
        <ValueComponent>
            <IfcMeasureValue>
                <IfcReal RealValue="0.0"/>
            </IfcMeasureValue>
        </ValueComponent>
        <ValueComponent>
            <IfcMeasureValue>
                <IfcReal RealValue="100.0"/>
            </IfcMeasureValue>
        </ValueComponent>
        <ValueComponent>
            <IfcMeasureValue>
                <IfcReal RealValue="130.0"/>
            </IfcMeasureValue>
        </ValueComponent>
        <ValueComponent>
            <IfcMeasureValue>
                <IfcReal RealValue="175.0"/>
            </IfcMeasureValue>
        </ValueComponent>

        [etc.]

    </DefiningValues>
    <DefinedValues>
        <ValueComponent>
            <IfcMeasureValue>
                <IfcReal RealValue="0.0"/>
            </IfcMeasureValue>
        </ValueComponent>
        <ValueComponent>
            <IfcMeasureValue>
                <IfcReal RealValue="160.0"/>
            </IfcMeasureValue>
        </ValueComponent>
```

```
            <ValueComponent>
                <IfcMeasureValue>
                    <IfcReal RealValue="400.0"/>
                </IfcMeasureValue>
            </ValueComponent>
            <ValueComponent>
                <IfcMeasureValue>
                    <IfcReal RealValue="1385.0"/>
                </IfcMeasureValue>
            </ValueComponent>

            [etc.]

        </DefinedValues>
    </TableProperty>

    <IfcSimpleProperty XMLID="i9003" Name="OriginalTestID">
        <ValueComponent>
            <IfcMeasureValue>
                <IfcString StringValue="SOFAF32"/>
            </IfcMeasureValue>
        </ValueComponent>
    </IfcSimpleProperty>
```

Note how the RelatedObjects attribute in the <IfcRelAssignsProperties> element has a value of "i261" which is the XMLID of the <IfcFurniture> element. The RelatingPropertyDefinition attribute in the<IfcRelAssignsProperties> element has a value of "i9000" which is the XMLID of the <IfcPropertySet> element. Finally, the HasProperties attribute of the <IfcPropertySet> element has the value "i9001 i9002 i9003" which correspond to the three elements that define the property set.

7.12 Escape routes

Section 7.6.3 suggests that the introduction of new property sets be considered to incorporate the C/AS1 definitions for escape routes. Part 3: Means of Escape of C/AS1 provides general principles and details of determining appropriate methods to allow occupants of a building to escape in the event of a fire. Figure 7-6 shows how escape routes are defined in C/AS1.

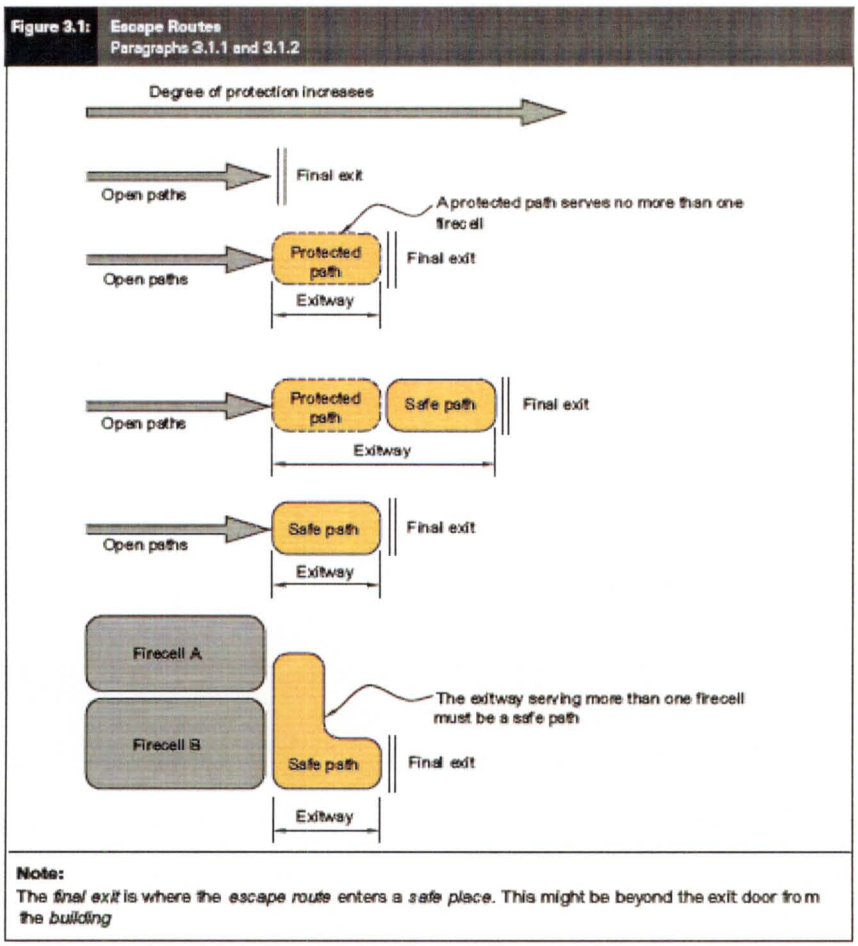


Figure 7-6. Escape routes as defined in C/AS1 (Figure 3.1 taken from BIA, 2001).

It is important to realise that the escape route property of a space is not inherent to a given type of space but determined from its function in a building. Whether a space forms part of a "*dead end*", an "*open path*" or a "*protected path*" would depend on its topological relationship to other escape route elements and the construction characteristics of its boundaries. A single space (e.g. a corridor) at an extreme of a building that forms an "*escape route*" would be a "*dead end*" and an "*open path*" which then could then be reduced to an "*open path*" if it then reaches a junction with other "*open path*" escape route elements. A space that has "*building elements*" that form "*smoke separations*" around the space boundary becomes a "*protected path*" and a space that has "*building elements*" that form "*fire separations*" around the space boundary becomes a "safe path". Furthermore, "*safe paths*" can include "*external walls*" or be a function of distance when exposed to open air. The creation of appropriate property sets in the IFC Model would require human intervention (or a

specifically created automated tool) to define which particular elements of an escape route meet a precise C/AS1 definition.

Chapter 8:
INTEGRATING THE IFC BUILDING PRODUCT MODEL WITH ZONE FIRE SIMULATION SOFTWARE

Spearpoint M J, Integrating the IFC building product model with zone fire simulation software. Proc. International Conference on Building Fire Safety, QUT, Gardens Point Campus, Brisbane, Australia, 20-21 November 2003, pp. 56-66.

Paper Referees:

One anonymous reviewer

8.1 Abstract

There is an international effort to develop a standardised, object-oriented "building product model" that overcomes the restrictions of conventional electronic descriptions of buildings and similar facilities. This product model is known as the Industry Foundation Class (or IFC) Model. The IFC Model consists of entities that represent a building; the properties needed to describe those entities and the inter-relationship between entities.

This paper briefly describes the structure and content of the IFC building product model and identifies those parts of the model that are relevant to fire engineering. The ability of zone fire simulation software to interpret IFC files is discussed and this is demonstrated using the **CFAST** and **BRANZFIRE** software. The interpretation of an IFC file is limited by the capabilities and restrictions of the fire simulation software as well as the IFC model.

8.2 Introduction

Fire simulation software (commonly referred to as 'fire models') are often used by fire engineers to develop and analyse design scenarios. In order to carry out the design process, a considerable amount of time can be spent specifying the input to the simulation software prior to execution. This information can be quite voluminous and needs to be accurately transferred to the simulation software. Items such as the geometry and topology of a building, the location of items within the building and the properties of those items are just some of the information that may be required. Mistakes and missing information when providing the input information can lead to inappropriate output from the simulation software.

Traditionally, such information might be entered by hand although the building plans may have been generated by Computer Aided Design (CAD) software. Even where there is the ability to directly transfer CAD data into fire simulation software there can be significant limitations. These limitations stem from a variety of reasons including where the fire simulation software tool may be incapable of interpreting any kind of CAD file or the CAD file does not adequately describe the building in a form that the software can extract the relevant details.

Ideally we would like to have a method that allows us to specify a building in a single format and then be able to transfer that description to any software tool we require with the maximum utilisation of the initial description and the minimum need to add additional information. Furthermore, we might want to append new information to the initial description of the building as a result of performing a simulation. Mowrer and Williamson (1998) proposed the integration of CAD information with fire specific properties of a building and its contents. The key features identified by Mowrer and Williamson required by CAD systems to permit integration included object-orientation, the association of attributes with objects and the ability to extract attributes from a CAD-developed drawing database.

8.3 Zone fire simulation software

It is not the purpose of this paper to provide a detailed description of zone fire simulation software or their underlying models. Quintiere (2002) provides a description of the basic conservation equations and relationships that are used by most zone fire simulation software. The atmosphere within a compartment is normally split into two zones; the hot gas layer as result of the fire and the cool layer below. The physical conditions within these layers are considered vertically and horizontally uniform. A survey by Olenick and Carpenter (2003) has shown that there are many such fire simulation software tools available. The software ranges in complexity in terms of the input requirements, the output capabilities and the extent of a fire hazard scenario represented.

Zone fire simulation software tools often make a number of assumptions or simplifications regarding the geometry of the spaces being modelled and the materials involved. For example, it is typical but not necessarily always the case to assume that spaces have a rectangular foot-print and that ceilings are smooth and horizontal. Assumptions such as these will need to be accounted for as we attempt to map the electronic description of a real building to the idealised view required by some zone fire simulation software. Some zone fire simulation software tools have a limited number of allowable spaces and/or a limited number of connections such as doors

between spaces. In the zone fire simulation software these connections might be assumed to be in the centre of the parent wall rather than at its actual location.

Although zone fire simulation tools all follow the same basic philosophy regarding the way in which the fire environment is represented, individual fire zone simulation tools may have components that are not present in other tools. For example, one tool may have a glass breaking algorithm which requires specific information not required by other tools that do not implement such an algorithm.

8.4 Product models

In general, any product can be considered to consist of a collection of 'entities'. A 'product model' expresses the type of entities that represent the product; the properties that are needed to describe those entities and the inter-relationship between entities. The description of a product model is commonly known as its 'schema'. A 'building product model' is a product model that specifically relates to buildings where entities may be physical objects such as doors, windows, walls etc. or more conceptual entities such as spaces or processes, contractual details etc. So for example, within a building product model a door entity has specific properties (such as its dimensions and construction material/s) and the relationship with the wall in which it is located (its position, orientation etc.).

Fire engineering is only one of the domains in the Architectural Engineering and Construction / Facilities Management (AEC/FM) industry that can benefit from a building product model (Chapter 4). Other domains might include architecture, structural engineering, environmental engineering, building services and many others. Many parameters related to a building are common to a range of disciplines including fire engineering. These parameters may include the building geometry and topology, the materials and components used in the construction and the location of the structure within the broad environment. However, due to the specialised needs of fire engineering, there are also parameters that are unique to the domain.

There are a number of issues to consider when examining the ability to describe a building in a product model and transfer that description to a simulation software tool.

We will find that all of the issues discussed in the paragraphs below related to the specification, implementation and interpretation of the IFC Model impact on the integration with zone fire simulation software.

8.4.1 Scope of the product model

Product models can be thought of being of two types; either they are general or they are domain specific (Ito, 1995). A general product model supports the generation and sharing of project data through the complete building lifecycle amongst a diverse range of domains. A general product model does not attempt to include every aspect of a product as this would likely be too complex and take too long to develop. Instead a general product model describes entity types at relatively high level.

Conversely, domain specific models retain as much of the project data as required for use within a domain. For the fire engineering domain we might find that a general product model does not provide all of the detail we need for fire simulation software tools or any other fire engineering related task. A domain specific product model could be created which has all those entities relevant to the fire engineering domain but this can lead to problems when sharing that project data with participants in other domains. There would need to be a number of software tools available to interpret each domain specific model. Furthermore, domains outside fire engineering might find the domain specific model has essential information missing or not in a form that is useable by them.

8.4.2 Implementation of the product model

Even where a product model completely describes a domain, we might find that specific software tools may only implement a reduced proportion of the whole product model. This can mean that although the product model has an entity described within its schema, the software tool cannot be used to create a specific instance of that entity or completely populate the properties associated with the entity. In some cases it may be possible to manually add entities in lieu of having an appropriate software tool.

8.4.3 Extraction of domain specific information

Finally the interpretation of the product model may lead to problems. The structure of entities may not be compatible with the specific requirements of the target software tool. This can happen where there might be insufficient detail in the product model as discussed above but also where the requirements of the software tool include simplifications and assumptions about a product that need to be accounted for during the transformation process.

8.4.4 Static and dynamic views

The IFC Model only provides a static view of a building whereas we may be interested in a dynamic system when we consider our fire simulation requirements. For example, we may wish to change the opening proportion of a door during the scenario or alter the flow paths along ducts during the operation of a smoke control system. In its current state, the IFC Model is unable to provide this type of dynamic information and the user would need to modify their input before executing the simulation software.

8.5 IFC Model

The IFC Model development began around 1996 and is an extension of a number of earlier projects. The latest version of the IFC Model is 2x (Liebich and Wix, 2000) which builds on the earlier 1.5 and 2.0 major releases of the model. In November 2002 the IFC 2x Model was accepted by the International Standards Organisation (ISO) as a Publicly Available Specification, ISO/PAS 16739.

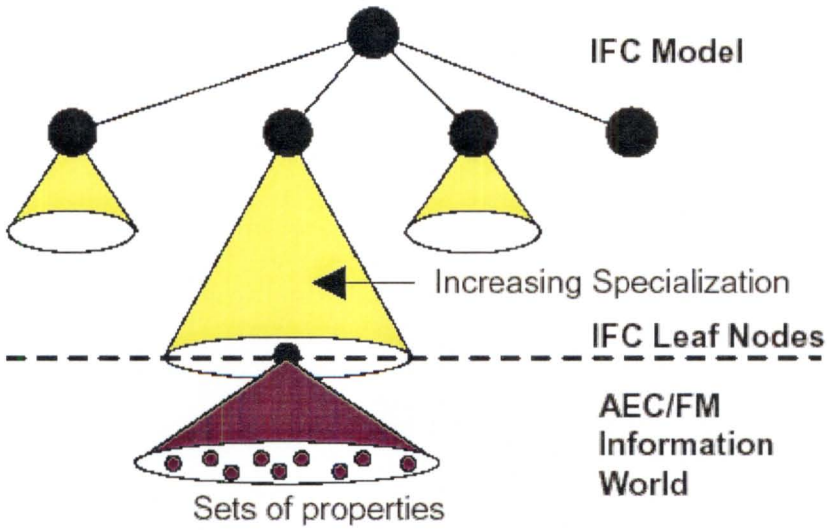


Figure 8-1. Limits of the IFC Model.

Primarily IFC files are exchanged using STEP (Standard for the Exchange of Product model data) technology, ISO 10303 (International Organization for Standardization, 2002) however more recently an equivalent format using XML (eXtensible Markup Language) has been made available referred to as 'ifcXML' (Liebich, 2001).

The IFC Model addresses the limited scope of a general product model using 'property set definitions' (PSD). The high level entities terminate at "leaf nodes" which allow object types to be extended using the PSD sub-schema (Figure 8-1, from Liebich and Wix, 2000). The specification of PSDs can be made outside of the main IFC Model by specialists within their domain. Even with the use of PSDs, there may still be specific object types missing or incomplete simply due to the fact that nobody has yet included the information in the product model. This is the case with many of the entities we might want to use as fire engineers. A review of the properties in the current IFC Model (Chapter 7) with respect to fire engineering found that whilst there are a number of fundamental material properties and several regulatory related properties in the IFC Model, there are also many areas in which the model can be extended.

Even in its current 'limited' form, the IFC Model is very complex with over 350 individual entities specified in the schema. In many cases building elements are described by a generic entity with specific elements defined as sub-types of that generic entity. For example, *IfcElement* is used as a generic entity of all components that make up an AEC product of which *IfcBuildingElement* is one type. The *IfcBuildingElement* entity is used to describe specific building elements such as walls, windows, stairs etc. using *IfcWall*, *IfcWindow* and *IfcStair* entities (Figure 8-2). Other sub-types of *IfcElement* such as *IfcOpeningElement* also have one or more sub-types.

Although IFC specifications had been available since late 1996, implementation activity was very limited because most companies were waiting to see what the large CAD vendors would implement. A number of organisations formed the BLIS (Building Lifecycle Interoperable Software) project (BLIS Project Companies, 2003) so that actual applications could be developed and demonstrated. Software tools included CAD, thermal design, quantity take-off, model consistency checker and others. The BLIS project used the then current IFC Release 2 model as its basis.

8.6 An ifcXML Parser

A general set of classes has been written in C++ in order to extract information relevant to fire simulation software tools. These classes have then been used to create an **ifcXML Parser** program. These classes were developed independent of any specific fire simulation tool, be that a zone, Computational Fluids Dynamics (CFD) or an evacuation simulation tool etc. The classes interpret the IFC Model into an intermediate set of data structures relevant to fire simulation software. Interfaces to specific models can then be written to translate the intermediate data structures into a format compatible with the appropriate fire simulation software. Finally the **ifcXML Parser** software is provided with a Windows-based user interface for ease of use (Figure 8-3).

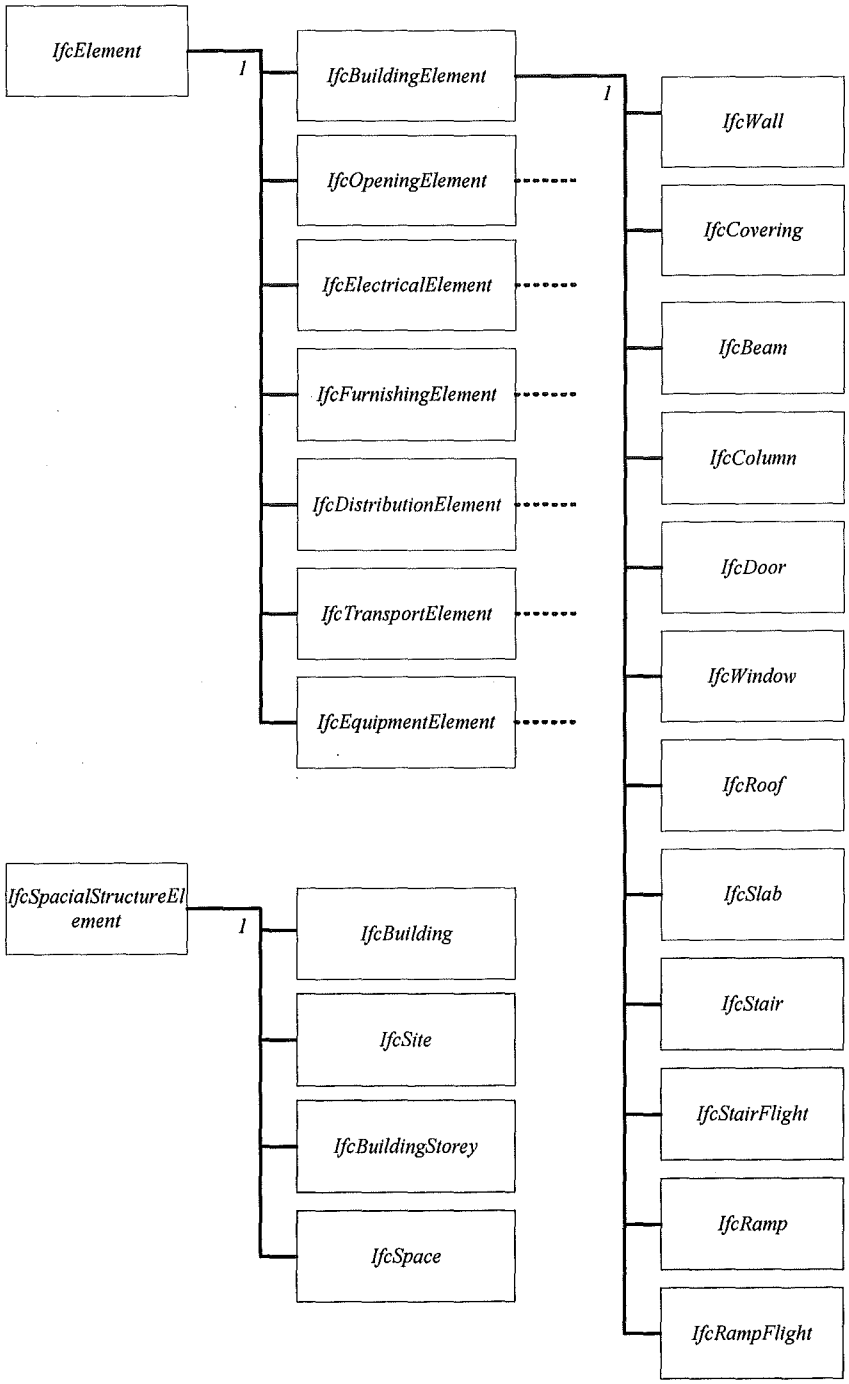


Figure 8-2. Examples of the relationship between generic and specific entities in the IFC Model.

Advantages of this approach are that should the IFC Model change then only the parser classes need be modified to reflect those changes without having to rewrite the interfaces to the intermediate data structures. Additional interfaces can be easily developed using the parser classes as their basis. The fire simulation specific

interfaces can also use PSD data supplied from an external resource to enhance the translation output as demonstrated in this paper.

8.7 Software tools

Two widely available multi-room fire zone simulation tools have been selected to demonstrate the integration with the IFC Model. The Fire And Smoke Transport (FAST) family (Bukowski et al., 1989) of zone fire simulation software (i.e. **HAZARDI**, **FAST**, **CFAST**, **FASTLite**) has been available from NIST for well over two decades. Many fire engineers are familiar with these software packages and they have been extensively used for a wide variety of analyses (Bukowski, 1996). The **BRANZFIRE** zone fire simulation software (Wade, 2002) is available from the Building Research Association of New Zealand (BRANZ) and is under continued development.

Microsoft **Visio Professional 2002** was chosen to create building descriptions and export them as IFC files. Microsoft **Visio** is an additional part of the Microsoft Office suite and an IFC "addin" is freely available from Microsoft. Microsoft **Visio** can only exchange IFC files in XML format and the IFC addin was one of the tools developed as part of the BLIS projects. Microsoft **Visio** is a 2D draughting design tool which includes templates for building core; wall, shell and structure and many other related elements. Other mainstream CAD software (such as **ArchiCAD**, **AutoCAD**, **Microstation**) have or are likely to have the ability to use IFC files.

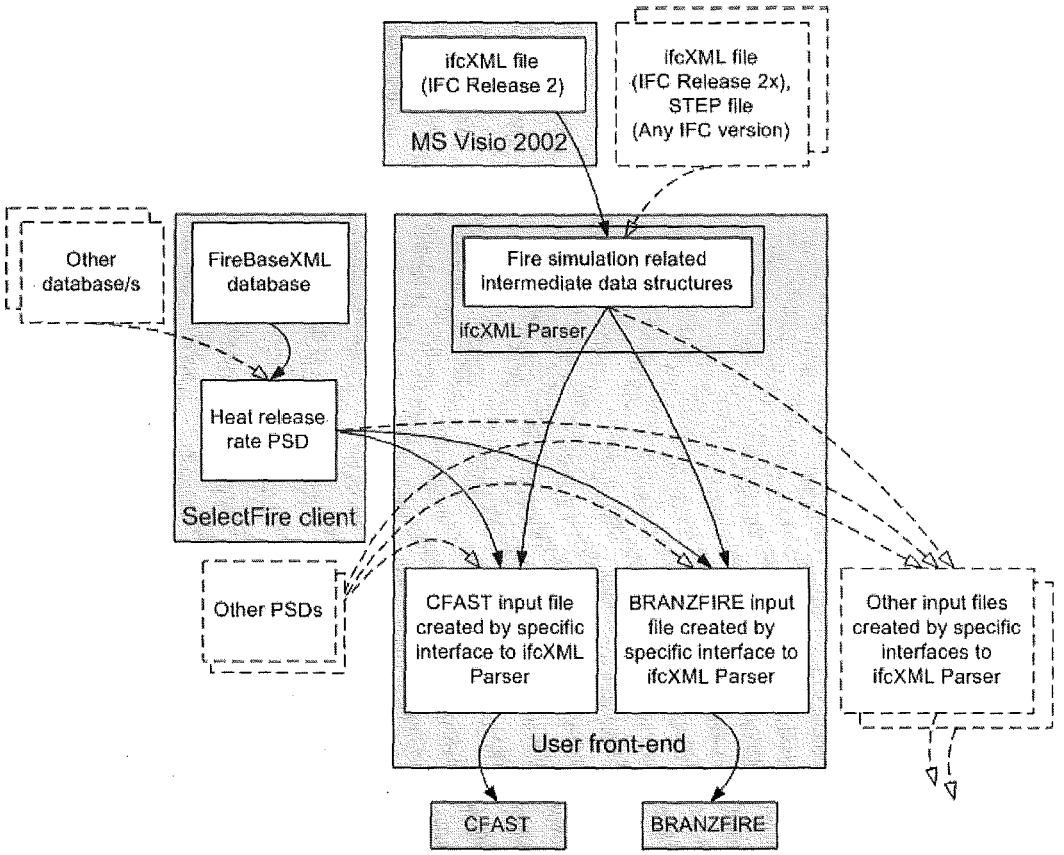


Figure 8-3. Data transfer between an IFC Model file and fire simulation software (dotted items not currently implemented).

8.8 Test buildings

Due to the complexity of the IFC Model it was impractical to develop the ifcXML Parser without using test case buildings as a means of verifying the effectiveness of the translation process. Primarily two relatively simple buildings were used as these test cases. The basic information we want to transfer from an IFC Model compatible CAD plan is the geometry of the spaces, connections between spaces and the material properties such items as wall linings etc. Both of these buildings highlighted a number of the issues related to the specification, implementation and interpretation of the IFC Model.

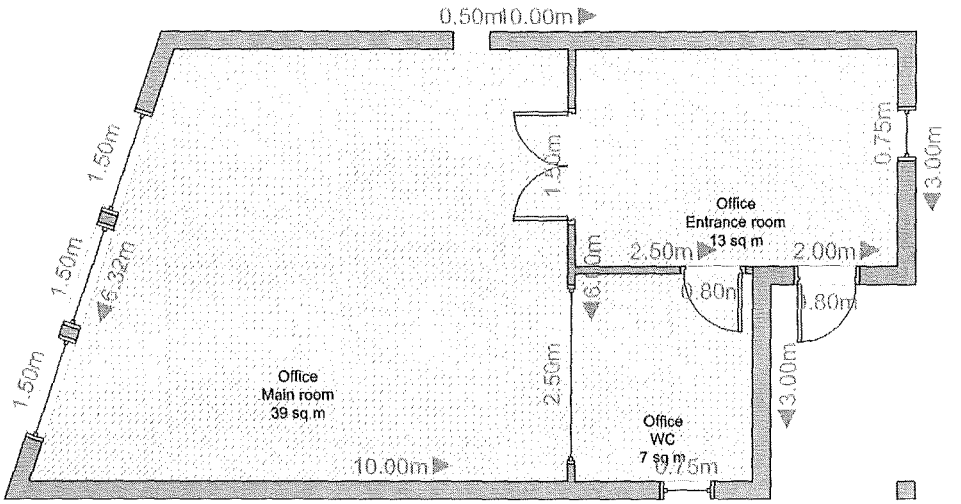
The first building was a 3-room, single storey structure that is available from the BLIS website (http://www.blis-project.org/BLIS_XML/simple2_001204.zip) as shown in Figure 8-4(a). The second test case was the ground floor of a two-storey family home

was directly draughted in Microsoft **Visio** shown in Figure 8-4(b). The building geometry was based on a house which has been used in a number of fire-related experimental projects (Spearpoint and Smithies, 1999). Several furniture items, taken directly from the Microsoft **Visio** furniture types, were arbitrarily added to the house in order to test some components of the **ifcXML Parser** software.

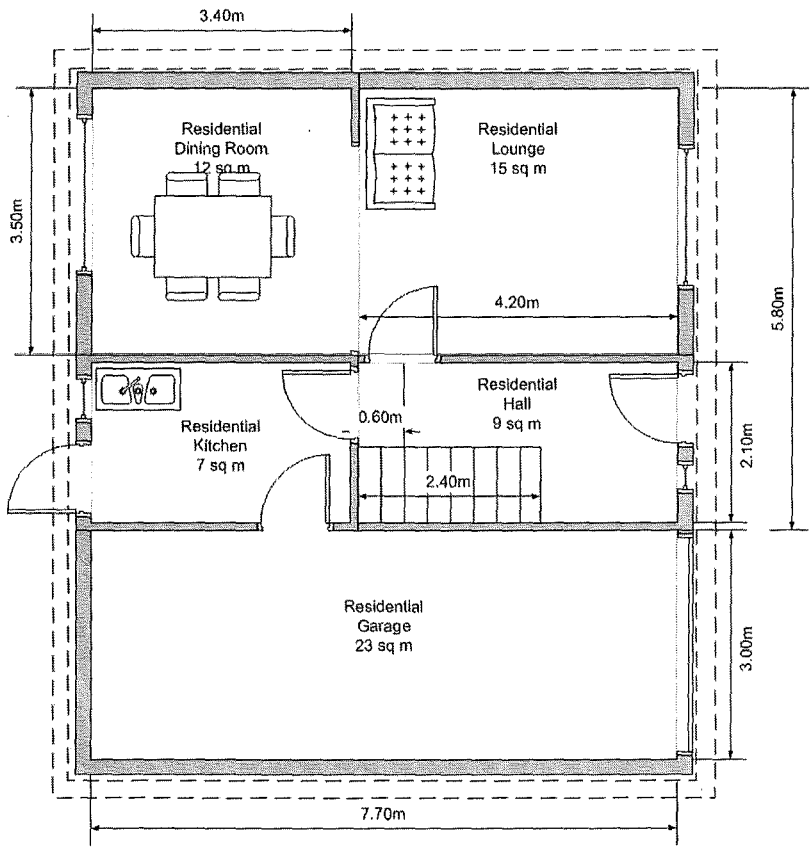
8.8.1 Geometry

In a zone fire simulation tool the basic geometry of a building is divided up into a collection of individual spaces, usually referred to as rooms or compartments. The structure of the IFC Model allows each individual space to be identified through the *IfcSpace* entity.

Using the **ifcXML Parser**, *IfcSpace* entities are mapped to zone fire simulation rooms with the *IfcSpace* entity label used to identify the name of the room in the fire simulation input file. The limitation of this mapping process is that spaces bounded by complex wall geometries will not convert well into a zone fire simulation tool that expects rectangular footprints. As shown in Figure 8-4(a), the simple2_001204 building includes a non-rectangular room. Since both **CFAST** and **BRANZFIRE** require rectangular-shaped rooms, the **ifcXML Parser** translates this room into a room of dimensions 7.45m by 6.00m. This limitation is an example of where the simplifications used by a target software tool mean that the complete details given by the IFC Model are lost in the translation.



(a)



(b)

Figure 8-4. Test buildings used to verify the IFC Model translation, (a) The simple2_001204 building available on the BLIS website imported into **Visio**, (b) Ground floor of the two-storey residential building drafted in **Visio**.

The height of a space is obtained from the height of the bounding box of the *IfcSpace* entity. Since Microsoft **Visio** is a 2D drafting tool the user cannot graphically designated the height of a space. Instead the height is specified through **Visio**'s space properties dialogue box.

8.8.2 Walls

A space is normally bounded by a wall, is open to a neighbouring space or open to the outside. Solid boundaries such as walls may be constructed of one or more layers each of different materials and thicknesses, for example, an outside wall might be brick but the internal partitions might be gypsum wallboard.

In the IFC Model *IfcSpace* entities are bounded by one or more *IfcSpaceBoundary* entities. These bounding entities can be *IfcWall* entities or some other appropriate entity. The **ifcXML Parser** identifies the *IfcWall* entities bounding a space. The thickness of the wall is obtained from the *IfcLayer* entity that is referenced by an *IfcWall* entity. An *IfcWall* entity can consist of several *IfcLayer* entities however the current version of the **ifcXML Parser** will only recognise the first *IfcLayer* entity for reasons discussed here.

There are several general issues associated with the translation of bounding entities from the IFC Model to **BRANZFIRE** and **CFAST**. An *IfcSpace* generally has more than one bounding wall and these are not necessarily defined as having similar construction. **BRANZFIRE** and **CFAST** both assume that a room has a single type of wall surrounding it and this leads to a number of inconsistencies when mapping between the IFC Model and the specific zone fire simulation tool.

The material property from an *IfcWall* entity is mapped to a **BRANZFIRE** or **CFAST** room wall material. The material is presently assumed to be in the 'Description' property of the *IfcWall* entity and should be one of the named properties available in the appropriate **BRANZFIRE** or **CFAST** thermal properties database. If the *IfcWall* entity description does not match a thermal database entry then the zone fire simulation may not execute as expected.

Where no wall bounding a space has a description then the equivalent **BRANZFIRE** and **CFAST** room walls will not be assigned a material. Where the walls have a mix of descriptions, the **BRANZFIRE** and **CFAST** wall material will be assigned the description of the 'last' wall that bounds the space. The 'last' bounding wall entity will depend on how **Visio** outputs its IFC file and thus cannot be necessarily predicted or assumed to be the same each time the file is saved. Finally, if some space boundary walls are given material descriptions and others not this may lead to the **CFAST** wall materials not correctly matching their associated rooms. This is due to way in which a **CFAST** input file is formatted.

Visio does not appear to have the facility to create multi-layer walls. Instead a single layer is used to specify the wall thickness and because of this limitation the current parser necessarily maps an *IfcWall* and its associated *IfcLayer* thickness to the **BRANZFIRE** wall lining and does not map any properties to the **BRANZFIRE** wall substrate.

8.8.3 Openings

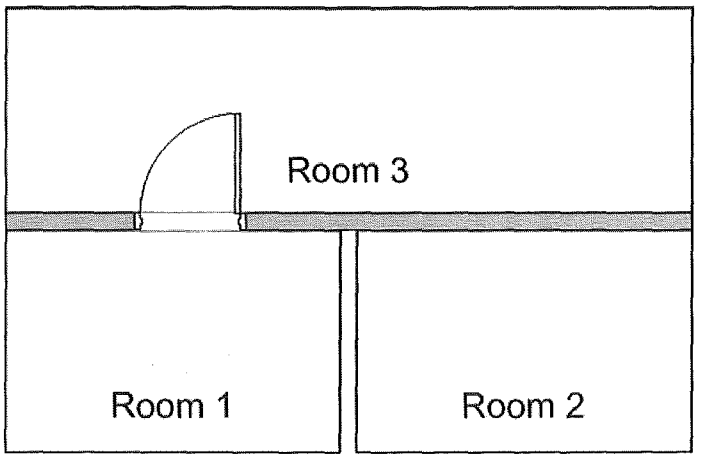
8.8.3.1 Opening types

The boundaries of a space typically contain openings that connect to neighbouring spaces or the outside. These openings might include doorways, windows, holes etc. In the case of doorways and windows, the opening may be completely filled by the door or the pane of glass. Alternatively there may be an open path through the opening if the door or window is partially or fully open.

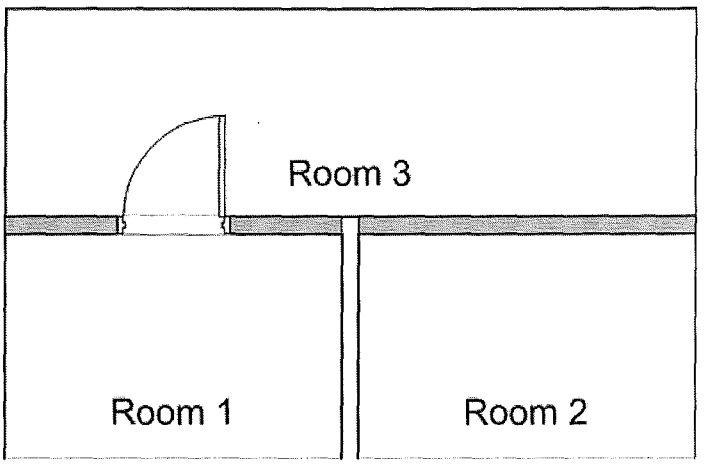
The IFC Model specifies openings in entities using voiding entities. This allows for the identification of the parent entity for a particular opening (e.g. in which wall do we find a particular door) and also the identification of a group of openings in a specific entity (e.g. which doors form openings in a specific wall). The **ifcXML Parser** program identifies all these openings associated with each space during the parsing of the *IfcSpace* entities and determines the appropriate connections. The user can specify whether they want all windows and/or doors to be fully open or completely closed however it is currently not possible to specify the states of individual openings.

8.8.3.2 Connecting spaces via openings

The simple2_001204 building allowed connections between spaces to be made by using the fact that an opening is referenced to by both parent spaces. However when a new building plan is created in **Visio** an opening only belongs to a single space making it impossible to identify the connecting space using an algorithm designed for files in the form of the simple2_001204 building. Instead a second algorithm was developed that uses the fact that we can identify the space boundary between two spaces and find if a wall belongs to that boundary. If a wall does exist on the boundary, we can check if any other entities such as doors and windows void the wall.



(a)



(b)

Figure 8-5. IFC Model translation of walls that bound common spaces, (a) single common wall, (b) individual walls, wall between Room 1 and Room 2 not shown.

This second algorithm has particular issues regarding bounding walls that are common to several spaces. If we consider the geometry shown in Figure 8-5(a) where the wall is common to Room 1 and Room 2 the **ifcXML Parser** would determine that Room 3 is next to both Room 1 and Room 2 via their common wall. When finding which two rooms the door connects, the **ifcXML Parser** would be unable to determine whether the door links Room 3 with Room 1 or Room 3 with Room 2

simply from knowing only that the rooms are next to each other. Currently the way to overcome this problem is to have separate walls bounding the smaller compartments such as shown in Figure 8-5(b) in which case the door can only connect Room 3 with Room 1 via the wall that only bounds these two rooms. An alternative solution would be to consider the relative coordinates of the door with respect to the connecting rooms in order to determine the appropriate link and this may be implemented in a future version of the **ifcXML Parser**.

What is not clear regarding this discrepancy between the simple2_001204 building and the house draughted in **Visio** is whether the implementation in **Visio** is intentionally different to the BLIS example. The current version of the **ifcXML Parser** assumes that there are no common walls between spaces so that floor plans appropriately drafted in **Visio** are correctly translated.

8.8.4 Floors and ceilings

In the IFC Model *IfcSpace* entities do not have an explicit associated ceiling or floor entity. Instead the floor and ceiling properties for each room are obtained from *IfcSlab* entities. It is assumed by the **ifcXML Parser** that a single slab defines the properties for the whole floor which means that it is not possible to have different properties for different rooms. The lack of ceiling and floor entities in the IFC Model presents difficulties where we might want to consider these independently from the slab. For example we might be interested in knowing the flammability characteristics of a floor system which might not be the same system in different spaces.

To define the floor and ceiling slabs, the user creates a **Visio** 'slab' from the 'Wall, shell and structure' stencil. The slab can be located anywhere on the plan and does not necessarily need to cover the whole floor plan. The user then needs to specify the 'Slab type' to indicate whether the slab is a 'Floor' or 'Roofslab'. The user can also specify the thickness of the slab and the material.

8.8.5 Fire

In order to conduct a simulation, the software needs to be able to determine the characteristics of the fire itself. Fundamentally this could be an ignition source and the thermo-physical properties of the fuels however many fire simulation tools are unable

to model the ignition and fire spread mechanisms particularly for mixed fuels in complex arrangements. As an alternative many zone fire simulation tools typically require the user to specify heat release rate data to define the fire. In many situations a fire in a building is the result of the burning of the contents and these contents are often items of furniture. The IFC Model includes a variety of furniture items and therefore these can be used as suitable fire sources.

It is not possible to directly specify the heat release rate of the contents of a building using **Visio** and this is likely to be the case with any other commercial CAD tool. Instead heat release rate data needs to be added from an external source and an IFC Model PSD heat release rate data format has already been defined in Chapter 7. The heat release rate data can be from any convenient source of which one such source is the FireBaseXML database (Chapter 5). A software tool called **SelectFire** has been developed to interrogate such a database is available and a translation has been written to convert a FireBaseXML record into a PSD.

Items of furniture without heat release rate data are parsed but not added to the **BRANZFIRE** or **CFAST** input files. With the translation to **CFAST**, if the IFC file includes more than one item of furniture with an attached heat release rate data then the 'last' entity found will be used. However with **BRANZFIRE**, if the IFC file includes more than one item of furniture with an attached heat release rate data then the **ifcXML Parser** adds each entity to the room. The 'first' furniture entity found will specify the location of the fire but the user can easily modify this in **BRANZFIRE** prior to running a simulation. In each case the 'first' and 'last' furniture entities will depend on how **Visio** outputs the IFC file and thus cannot be necessarily predicted or assumed to be the same each time the file is saved.

BRANZFIRE expects the energy output from the fire to be expressed in kW and the time in seconds as supplied by FireBaseXML. **CFAST** expects the energy output from the fire to be expressed in Watts so the PSD must be appropriately interpreted during translation. Also **CFAST** only allows 20 data points in its heat release rate curve and if more points are specified then only the first 20 will be used.

Where appropriate, the fire is positioned in the room using the location provided in the IFC file. The dimensions of the fire are taken from the bounding box of the *IfcFurniture* entity. For complex shapes this may only be an approximate representation. The user would need to edit the simulation input if they wanted the dimensions of the fire to change over time, for example, if the area of the fire changed with time. This is one example of where the IFC Model is currently unable to deal with dynamic views of a building.

8.9 Future work

The complexity and scope of the IFC Model means that there are still considerable enhancements that can be added to the **ifcXML Parser**. For example, at the time of writing the translation to the zone fire simulation software does not fully process *IfcStairs* entities. It should also be possible to allow the user to specify the opening state of individual windows or doors through **Visio**. There is also the need to test additional buildings such as multi-storey structures and buildings that have been draughted in other IFC Model compatible software where the files are either in STEP or ifcXML format.

The current IFC Model has entities that can be used to specify components of smoke control systems such as fans and dampers however there may be a need to enhance these entities so as to encompass all of the available components and to describe all relevant properties. In the longer term the core IFC Model could be extended to include specific fire safety systems such as detection components, suppression systems etc. Some of this could be achieved using the PSD capabilities included in the IFC Model.

Future interfaces to other fire simulation software are possible and these might include CFD fire simulation software or people movement simulation tools. It might also be desirable to make the translation of an IFC Model file integral with the fire simulation software rather relying on a separate parser tool. This would need access to the fire simulation software's source program and may require considerable effort in order to integrate the parsing processes and a suitable user interface within the fire simulation software.

8.10 Conclusions

The work described in this paper demonstrates how the IFC Model is a general way in which to describe buildings and the means of using this model to enable the sharing of the description to fire zone simulation software. There is still much work to do including the expansion of the IFC Model to include more fire-specific information either in the core model or by using PSDs, enhancing the interpretation of the IFC files for fire zone simulation software and widening the scope of models that can use the IFC descriptions.

Chapter 9:
REVIEW OF FIRE PROTECTION ENGINEERING ENTITIES IN IFC 2X2

A reduced version of this chapter was provided as feedback to the IAI Modelling Support Group during January 2004.

9.1 Fire protection entities in IFC 2x2

The IFC 2x Edition 2 (IFC 2x2) model has significantly greater support for fire (protection) engineering than the previous IFC 2x Model. This section reviews the major additions to the IFC 2x2 and highlights any changes identified between IFC 2x and IFC 2x2. In particular, IFC 2x2 has a specific domain referred to as the *IfcPlumbingFireProtectionDomain* which covers aspects of fire extinguishing systems. However, there are many other entities and properties that are relevant to fire engineering scattered throughout the IFC 2x2 Model and these are described in the following sub-sections. All of the figures shown in this section are taken or adapted from those given in the IFC 2x2 documentation (Adachi et al., 2003). A copy of the IFC 2x2 architecture diagram is shown in Figure 9-1 (Liebich, 2003).

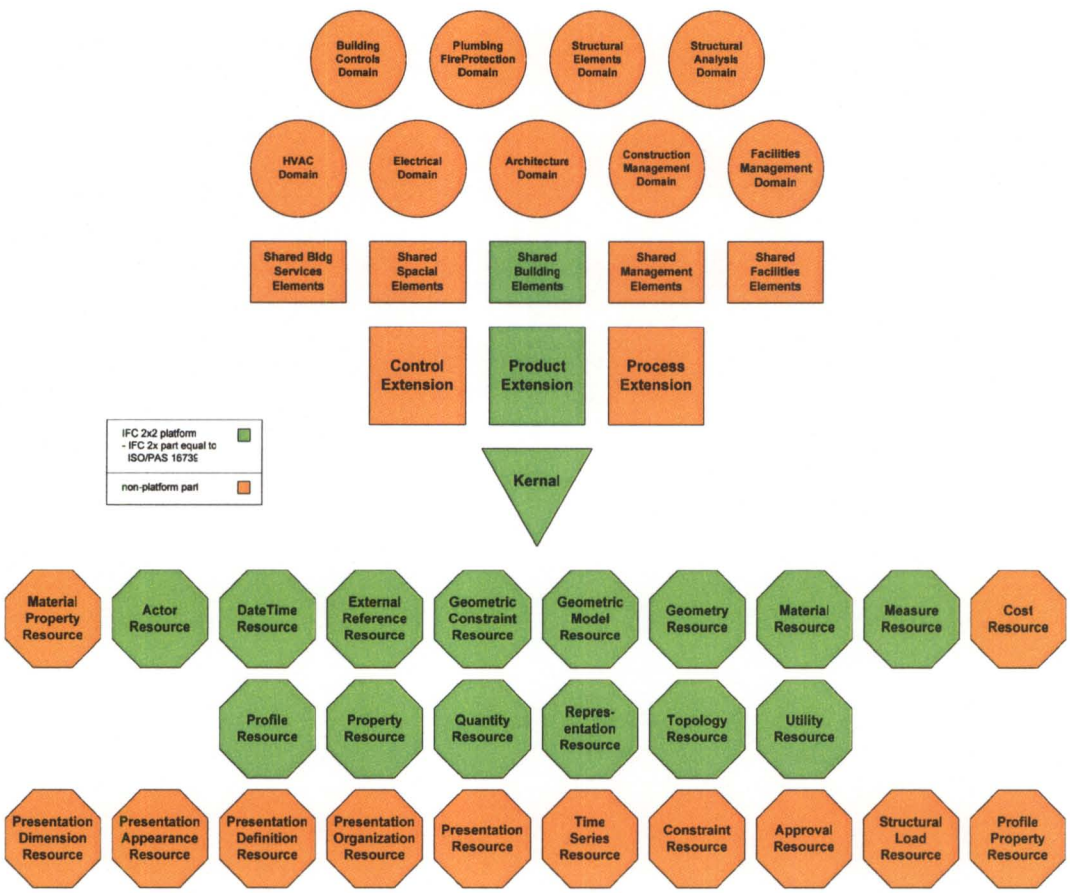


Figure 9-1. The IFC 2x2 architecture diagram.

9.2 Buildings and spaces

The IFC Model has entities that relate to buildings as a whole and the spaces that form those buildings. Several of these entities have properties that relate directly to fire protection engineering. The IFC 2x2 release has a number of changes to these entities in terms of fire-related properties and these are discussed below.

One significant change to IFC 2x2 over IFC 2x is that *Pset_SpaceCommon* no longer defines any directly fire-related properties but instead they are specified in *Pset_SpaceFireSafetyRequirements* (Figure 9-2). The properties consider the use of the space through the *MainFireUse*, *AncillaryFireUse* and *FlammableStorage* properties; associated risks and hazards with the *FireRiskFactor* and *FireHazardFactor* properties (also refer to Section 9.7.2); provisions for means of escape with the *FireExit* property (also refer to Section 9.3.1) and any active fire protection systems in the space by the use of the *SprinklerProtection*, *SprinklerProtectionAutomatic* (also refer to Section 9.5) and *AirPressurization* properties.

PropertySet Name	*Pset_SpaceFireSafetyRequirements*
Applicability	*IfcSpace*, *IfcZone* entity.
Applicable Classes	*IfcSpace*, *IfcZone*
Applicable Type Value	
Definition	Definition from IAI: Properties related to fire protection of spaces that apply to the occurrences of *IfcSpace* or *IfcZone*.

Property Definitions:

Name	Property Type	Data Type	Definition
MainFireUse	*IfcPropertySingleValue*	*IfcLabel*	Main fire use for the space which is assigned from the fire use classification table as given by the relevant national building code.
AncillaryFireUse	*IfcPropertySingleValue*	*IfcLabel*	Ancillary fire use for the space which is assigned from the fire use classification table as given by the relevant national building code.
FireRiskFactor	*IfcPropertySingleValue*	*IfcLabel*	Fire Risk factor assigned to the space according to local building regulations.
FireHazardFactor	*IfcPropertySingleValue*	*IfcLabel*	Fire hazard code of the space. The coding depends on the national fire safety regulations.

FlammableStorage	*IfcPropertySingleValue*	*IfcBoolean*	Indication whether the space is intended to serve as a storage of flammable material (which is regarded as such by the presiding building code. (TRUE) indicates yes, (FALSE) otherwise.
FireExit	*IfcPropertySingleValue*	*IfcBoolean*	Indication whether this object is designed to serve as an exit in the case of fire (TRUE) or not (FALSE). Here whether the space (in case of e.g., a corridor) is designed to serve as an exit space, e.g., for fire escape purposes.
SprinklerProtection	*IfcPropertySingleValue*	*IfcBoolean*	Indication whether the space is sprinkler protected (true) or not (false).
SprinklerProtectionAutomatic	*IfcPropertySingleValue*	*IfcBoolean*	Indication whether the space has an automatic sprinkler protection (true) or not (false). It should only be given, if the property "SprinklerProtection" is set to TRUE.
AirPressurization	*IfcPropertySingleValue*	*IfcBoolean*	Indication whether the space is required to have pressurized air (TRUE) or not (FALSE).

Figure 9-2. The IFC 2x2 *Pset_SpaceFireSafetyRequirements* property set

As discussed in Section 7.6.2, the New Zealand Acceptable Solution C/AS1 (BIA, 2001) defines the "*fire hazard category*" of a space. In IFC 2x there was no distinct property associated with an *IfcSpace* entity that would directly serve the purpose of declaring the "*fire hazard category*" however in IFC 2x2 the *FireHazardFactor* could be an appropriate property in which to hold this information. Similarly, IFC 2x2 includes the *FireExit* property that could be used to designate escape route elements defined by C/AS1. However, as discussed in Section 7.12, C/AS1 has a number of definitions for particular escape route elements that would not be solely met by the use of the *FireExit* property.

Table 9-1 shows other fire-related properties associated with buildings and spaces. These property sets are all defined at the core extension level (*IfcProductExtension*) of the IFC Model. As well as the *IfcSpace* and *IfcZone* entities having fire-related properties (as shown in Figure 9-2), similar properties are associated with the *IfcBuilding* and *IfcBuildingStorey* entities. The only property that is not already found

in the *Pset_SpaceFireSafetyRequirements* property set is the *ProtectedOpening* property associated with the *Pset_OpeningElement* property set.

Property set definition and associated IFC entity	Fire specific property name	Data type	Definition
Pset_BuildingCommon (*IfcBuilding*)	*MainFireUse*	*IfcLabel*	Main fire use for the building which is assigned from the fire use classification table as given by the relevant national building code.
	AncillaryFireUse	*IfcLabel*	Ancillary fire use for the building which is assigned from the fire use classification table as given by the relevant national building code.
	SprinklerProtection	*IfcBoolean*	Indication whether this object is sprinkler protected (true) or not (false).
	SprinklerProtection Automatic	*IfcBoolean*	Indication whether this object has an automatic sprinkler protection (true) or not (false). It should only be given, if the property "Sprinkler Protection" is set to TRUE.
Pset_Building StoreyCommon (*IfcBuildingStorey*)	*SprinklerProtection*	*IfcBoolean*	Indication whether this object is sprinkler protected (true) or not (false).
	SprinklerProtection Automatic	*IfcBoolean*	Indication whether this object has an automatic sprinkler protection (true) or not (false). It should only be given, if the property "Sprinkler Protection" is set to TRUE.
Pset_OpeningElement (*IfcOpeningElement*)	*FireExit*	*IfcBoolean*	Indication whether this object is designed to serve as an exit in the case of fire (TRUE) or not (FALSE). Here whether the space (in case of e.g., a corridor) is designed to serve as an exit space, e.g., for fire escape purposes.
	ProtectedOpening	*IfcBoolean* • Default Value: FALSE	Indication whether the opening is considered to be protected under fire safety considerations. If (TRUE) it counts as a protected opening under the applicable building code, (FALSE) otherwise.
Pset_TransportElement Common (*IfcTransportElement*)	*FireExit*	*IfcBoolean*	Indication whether this object is designed to serve as an exit in the case of fire (TRUE) or not (FALSE). Here whether the transport element (in case of e.g., a lift) is designed to serve as a fire exit, e.g., for fire escape purposes.

Table 9-1. IFC 2x2 fire-related properties for buildings and spaces.

In addition to the means of escape provisions specified in the property sets associated with buildings and spaces, the IFC 2x2 also includes properties that provide the occupancy characteristics of spaces (Table 9-2). These properties are likely to be useful when assessing whether means of escape meets regulatory requirements or where computer evacuation modelling is desired.

Property set definition and associated IFC entity	Fire specific property name	Data type	Definition
Pset_SpaceProgram Common (IfcSpaceProgram)	*OccupancyType*	*IfcLabel*	Occupancy type for this object. It is defined according to the presiding national building code.
	OccupancyNumber	*IfcCount Measure*	Maximum number of occupants for the designed usage of the space.
Pset_BuildingCommon (IfcBuilding)	*OccupancyType*	*IfcLabel*	Occupancy type for this object. It is defined according to the presiding national building code.
Pset_SpaceCommon (IfcSpace)	*OccupancyType*	*IfcLabel*	Occupancy type for this object. It is defined according to the presiding national building code.
	OccupancyNumber	*IfcCount Measure*	Maximum number of occupants for the designed usage of the space.
	HandicapAccessible	*IfcBoolean*	Indication whether this space (in case of e.g., a toilet) is designed to serve as an accessible space for handicapped people, e.g., for a public toilet (TRUE) or not (FALSE). This information is often used to declare the need for access for the disabled and for special design requirements of this space.
Pset_ZoneCommon (IfcZone)	*OccupancyType*	*IfcLabel*	Occupancy type for this object. It is defined according to the presiding national building code.
	OccupancyNumber	*IfcCount Measure*	Maximum number of occupants for the designed usage of the space.
	HandicapAccessible	*IfcBoolean*	Indication whether this space (in case of e.g., a toilet) is designed to serve as an accessible space for handicapped people, e.g., for a public toilet (TRUE) or not (FALSE). This information is often used to declare the need for access for the disabled and for special design requirements of this space.

Table 9-2. IFC 2x2 occupancy characteristic properties.

9.3 Passive fire protection

Several passive fire protection features were identified in IFC 2x associated with doors, walls, windows etc. These features have been retained in IFC 2x2 with several modifications as reviewed here.

9.3.1 Structural elements

Table 9-3 shows the fire-related properties for structural elements defined in IFC 2x2. IFC 2x generally only considered the fire resistance performance of structural elements by the use of the *FireRating* property. A comparison of the structural element property sets in IFC 2x2 with the equivalent ones in IFC 2x shows that almost all of the property sets have had additional properties included. These new properties identify means of escape provisions through the *FireExit* property (also specified for selected property sets associated with buildings and spaces, Table 9-1); fire and smoke containment with the *SmokeStop*, *SelfClosing* and *Compartmentation* properties and the performance of materials with the *Combustible* and *SurfaceSpreadOfFlame* properties (also refer to Section 9.7.1).

Property set definition and associated IFC entity	Fire specific property name	Data type	Definition
Pset_BeamCommon (*IfcBeam*)	*FireRating*	*IfcLabel*	Fire rating for this object. It is given according to the national fire safety classification.
Pset_ColumnCommon (*IfcColumn*)	*FireRating*	*IfcLabel*	Fire rating for this object. It is given according to the national fire safety classification.
Pset_CurtainWallCommon (*IfcCurtainWall*)	*FireRating*	*IfcLabel*	Fire rating given according to the national fire safety classification.
	Combustible	*IfcBoolean*	Indication whether the object is made from combustible material (TRUE) or not (FALSE).
	SurfaceSpread OfFlame	*IfcLabel*	Indication on how the flames spread around the surface, It is given according to the national building code that governs the fire behaviour for materials.
Pset_DoorCommon (*IfcDoor*)	*FireRating*	*IfcLabel*	Fire rating for this object. It is given according to the national fire safety classification.
	FireExit	*IfcBoolean*	Indication whether this object is designed to serve as an exit in the case of fire (TRUE) or not (FALSE). Here it defines an exit door in accordance to the national building code.

	SelfClosing	*IfcBoolean*	Indication whether this object is designed to close automatically after use (TRUE) or not (FALSE).
	SmokeStop	*IfcBoolean*	Indication whether the object is designed to provide a smoke stop (TRUE) or not (FALSE).
Pset_RampCommon (IfcRamp)	*FireRating*	*IfcLabel*	Fire rating for this object. It is given according to the national fire safety classification.
	FireExit	*IfcBoolean* • Default Value: FALSE	Indication whether this object is designed to serve as an exit in the case of fire (TRUE) or not (FALSE). Here it defines an exit ramp in accordance to the national building code.
Pset_RoofCommon (IfcRoof)	*FireRating*	*IfcLabel*	Fire rating for this object. It is given according to the national fire safety classification.
Pset_SlabCommon (IfcSlab)	*FireRating*	*IfcLabel*	Fire rating given according to the national fire safety classification.
	Combustible	*IfcBoolean*	Indication whether the object is made from combustible material (TRUE) or not (FALSE).
	SurfaceSpread OfFlame	*IfcLabel*	Indication on how the flames spread around the surface, It is given according to the national building code that governs the fire behaviour for materials.
	Compartmentation	*IfcBoolean*	Indication whether the object is designed to serve as a fire compartmentation (TRUE) or not (FALSE).
Pset_StairCommon (IfcStair)	*FireRating*	*IfcLabel*	Fire rating for this object. It is given according to the national fire safety classification.
	FireExit	*IfcBoolean* • Default Value: FALSE	Indication whether this object is designed to serve as an exit in the case of fire (TRUE) or not (FALSE). Here it defines an exit stair in accordance to the national building code.
Pset_WallCommon (IfcWall, IfcWallStandardCase)	*FireRating*	*IfcLabel*	Fire rating given according to the national fire safety classification.
	Combustible	*IfcBoolean*	Indication whether the object is made from combustible material (TRUE) or not (FALSE).
	SurfaceSpread OfFlame	*IfcLabel*	Indication on how the flames spread around the surface, It is given according to the national building code that governs the fire behaviour for materials.
	Compartmentation	*IfcBoolean*	Indication whether the object is designed to serve as a fire compartmentation (TRUE) or not (FALSE).

Pset_WindowCommon (*IfcWindow*)	*FireRating*	*IfcLabel*	Fire rating for this object. It is given according to the national fire safety classification.
	SmokeStop	*IfcBoolean*	Indication whether the object is designed to provide a smoke stop (TRUE) or not (FALSE).

Table 9-3. IFC 2x2 fire-related properties for structural elements.

As already discussed in Section 7.6.1, the use of an *IfcString* data type for the *FireRating* property allows FRR can be specified in the IFC Model the same form as used in the New Zealand Acceptable Solution C/AS1 (BIA, 2001). However, having the FRR expressed in numerical form would allow the more direct use of the information in calculations. Furthermore the numerical approach would allow an association of units with the FRR value since although C/AS1 uses minutes as its defined unit there can be cases where hours are used to describe an FRR. In order to use a numerical approach to storing this property the IFC Model would need some way in which the three components of the FRR specification could be stored. This would likely require a *FireRating* property that included an ordered list of integer values or specific named sub-properties that define the "*stability*", "*integrity*" and "*insulation*" values.

Additional properties which are likely to be useful in the assessment of means of escape provisions are specified for selected structural elements as shown in Table 9-4.

Property set definition and associated IFC entity	**Fire specific property name**	**Data type**	**Definition**
Pset_DoorCommon (*IfcDoor*)	*HandicapAccessible*	*IfcBoolean*	Indication that this object is designed to be accessible by the handicapped. It is giving according to the requirements of the national building code.
Pset_RailingCommon (*IfcRailing*)	*Height*	*IfcPositive LengthMeasure* / LENGTH UNIT	Height of the object. It is the upper height of the railing about the floor or stair. The size information is provided in addition to the shape representation and the geometric parameters used within.

Pset_RampCommon (IfcRamp)	*RequiredHeadroom*	*IfcPositive LengthMeasure* / LENGTH UNIT	Required headroom clearance for the passageway according to the applicable building code or additional requirements.
	RequiredSlope	*IfcPlaneAngle Measure* / PLANEANGLE UNIT	Required sloping angle of the object - relative to horizontal (0.0 degrees). Required maximum slope for the passageway according to the applicable building code or additional requirements.
	HandicapAccessible	*IfcBoolean* • Default Value: FALSE	Indication that this object is designed to be accessible by the handicapped. Set to (TRUE) if this ramp is rated as handicap accessible according the local building codes, otherwise (FALSE).
Pset_RampFlightCommon (IfcRampFlight)	*Headroom*	*IfcPositive LengthMeasure* / LENGTH UNIT	Actual headroom clearance for the passageway according to the current design. The shape information is provided in addition to the shape representation and the geometric parameters used within.
	Slope	*IfcPlane AngleMeasure* / PLANEANGLE UNIT	Sloping angle of the object - relative to horizontal (0.0 degrees). Actual maximum slope for the passageway according to the current design. The shape information is provided in addition to the shape representation and the geometric parameters used within.
Pset_StairCommon (IfcRamp)	*RequiredHeadroom*	*IfcPositive LengthMeasure* / LENGTH UNIT	Required headroom clearance for the passageway according to the applicable building code or additional requirements.
	HandicapAccessible	*IfcBoolean* • Default Value: FALSE	Indication that this object is designed to be accessible by the handicapped. Set to (TRUE) if this stair is rated as handicap accessible according the local building codes, otherwise (FALSE). Accessibility maybe provided by additional means.

Pset_StairFlightCommon (IfcStairFlight)	*Headroom*	*IfcPositive LengthMeasure* / LENGTH UNIT	Actual headroom clearance for the passageway according to the current design. The shape information is provided in addition to the shape representation and the geometric parameters used within.
	WalkingLineOffset	*IfcPositive LengthMeasure* / LENGTH UNIT	Offset of the walking line from the inner side of the flight. Note: the walking line may have its own shape representation
	WaistThickness	*IfcPositive LengthMeasure* / LENGTH UNIT	Minimum thickness of the stair flight, measured perpendicular to the slope of the flight to the inner corner of riser and tread.
	TreadLengthAtOffset	*IfcPositive LengthMeasure* / LENGTH UNIT	Length of treads at the walking line. Walking line position is given by the 'WalkingLineOffset'. The resulting value should normally be identical with *IfcStairFlight.TreadLength*, it may be given in addition, if the walking line offset for building code calculations is different from that used in design.
	TreadLength AtInnerSide	*IfcPositive LengthMeasure* / LENGTH UNIT	Minimum length of treads at the inner side of the winder. Only relevant in case of winding flights, for straight flights it is identical with *IfcStairFlight.TreadLength.*

Table 9-4. IFC 2x2 additional means of escape related properties for structural elements.

9.3.2 Dampers

Dampers are defined in the IFC 2x2 *IfcHvacDomain* and in terms of fire engineering, they contribute to the fire resistance rating and smoke containment of a structure. IFC 2x included properties for three types of fire-related damper all used by an *IfcDamper* entity. In IFC 2x2, dampers are specified using the *IfcDamperType* entity which is a sub-type of the *IfcFlowControllerType* entity. The same three types of fire-related damper are available in IFC 2x2 as in IFC 2x as described in the IFC 2x2 documentation (Adachi et al., 2003):

- *FIREDAMPER*: Fire damper used to prevent the spread of fire for a specified duration.
- *SMOKEDAMPER*: Smoke damper used to prevent the spread of smoke.
- *FIRESMOKEDAMPER*: Combination fire and smoke damper used to prevent the spread of fire and smoke.

PropertySet Name	*Pset_DamperTypeFire*
Applicability	Entity specific property set
Applicable Classes	*IfcDamperType*
Applicable Type Value	*IfcDamperType.PredefinedType = FIRE* or *IfcDamperType.PredefinedType = FIRESMOKEDAMPER*
Definition	Definition from IAI: Fire damper type attributes.

Property Definitions:

Name	Property Type	Data Type	Definition
ActuationType	*IfcPropertyEnumeratedValue*	*PEnum_FireDamper ActuationType* • GRAVITY • SPRING • OTHER • NOTKNOWN • UNSET	Enumeration that identifies the different types of dampers
ClosureRatingEnum	*IfcPropertyEnumeratedValue*	*PEnum_FireDamper ClosureRating* • DYNAMIC • STATIC • OTHER • NOTKNOWN • UNSET	Enumeration that identifies the closure rating for the damper
FireResistanceRating	*IfcPropertySingleValue*	*IfcLabel*	Measure of the fire resistance rating in hours (e.g., 1.5 hours, 2 hours, etc.).
FusibleLink Temperature	*IfcPropertySingleValue*	*IfcThermodynamic TemperatureMeasure*	The temperature that the fusible link melts.

Figure 9-3. The IFC 2x2 *Pset_DamperTypeFire* property set.

PropertySet Name	*Pset_DamperTypeSmoke*
Applicability	Entity specific property set
Applicable Classes	*IfcDamperType*
Applicable Type Value	*IfcDamperType.PredefinedType = SMOKE* or *IfcDamperType.PredefinedType = FIRESMOKEDAMPER*
Definition	Definition from IAI: Smoke damper type attributes.

Property Definitions:

Name	Property Type	Data Type	Definition
ControlType	*IfcPropertySingle Value*	*IfcString*	The type of control used to operate the damper (e.g., Open/Closed Indicator, Resetable Temperature Sensor, Temperature Override, etc.)

Figure 9-4. The IFC 2x2 *Pset_DamperTypeSmoke* property set.

It was noted that in IFC 2x the FireRating property for dampers was of type *IfcReal* which meant it could not hold the "*x/x/x*" style "*FRR*" classification required in New Zealand Approved Document Acceptable Solution C/AS1 (BIA, 2001). In IFC 2x2, the data type for the *FireRating* property has been changed to *IfcLabel* which overcomes this limitation.

9.4 Building services

IFC 2x2 now includes a general property set *Pset_FireRatingProperties* (Figure 9-5) in the *IfcSharedBldgServiceElements* interoperability layer. It is interesting to note that this property set includes a *FireResistanceRating* property which appears to fulfil the same function as the *FireRating* property used for the fire resistance of structural elements (e.g. *Pset_WallCommon* in Table 9-3). Similarly, the *IsCombustible* property in *Pset_FireRatingProperties* appears to fulfil the same function as the *Combustible* property for structural elements. It would seem appropriate to give these properties consistent names in any future revisions to the IFC Model as is the case with the *SurfaceSpreadOfFlame* property.

PropertySet Name	*Pset_FireRatingProperties*
Applicability	General property set
Applicable Classes	
Applicable Type Value	
Definition	Definition from IAI: Properties related to the combustion of materials for purposes of assessing fire hazard.

Property Definitions:

Name	Property Type	Data Type	Definition
FireResistanceRating	*IfcPropertySingleValue*	*IfcLabel*	Fire rating identifying the entity's fire resistive value (e.g., 1-hour, 2-hour, etc.) so that its resistance to fire can be compared to that of the surrounding structure.
IsCombustible	*IfcPropertySingleValue*	*IfcBoolean*	Combustibility (YES it is combustible or NO it is not combustible).
SurfaceSpreadOfFlame	*IfcPropertySingleValue*	*IfcLabel*	Surface spread of flame characteristics.

Figure 9-5. The IFC 2x2 *Pset_FireRatingProperties* property set.

9.5 Fire extinguishing systems

The *IfcPlumbingFireProtectionDomain* contains several property sets related to fire extinguishing systems through the *IfcFireSuppressionTerminalType* entity. The IFC 2x2 documentation (Adachi et al., 2003) defines the *IfcFireSuppressionTerminalType* as a "particular type of *IfcFlowTerminal* that has the purpose of delivering a fluid (gas or liquid) that will suppress a fire" and includes all forms of sprinkler, spreader and other form of terminal that is connected to a pipe work system. The available types of *IfcFireSuppressionTerminalType* are specified in the *IfcFireSuppressionTerminal-TypeEnum* entity shown in Figure 9-6 (*USERDEFINED* and *NOTDEFINED* excluded).

Value	*Definition*
BREECHINGINLET	Symmetrical pipe fitting that unites two or more inlets into a single pipe (BS6100 330 114 adapted)
FIREHYDRANT	Device, fitted to a pipe, through which a temporary supply of water may be provided (BS6100 330 6107)
HOSEREEL	A supporting framework on which a hose may be wound (BS6100 155 8201)
SPRINKLER	Device for sprinkling water from a pipe under pressure over an area (BS6100 100 3432)
SPRINKLERDEFLECTOR	Device attached to a sprinkler to deflect the water flow into a spread pattern to cover the required area (IAI)

Figure 9-6. IFC 2x2 values and definitions for the *IfcFireSuppression TerminalTypeEnum* entity.

The four property sets associated with the *IfcFireSuppressionTerminalType* are:

- *Pset_FireSuppressionTerminalTypeBreechingInlet*
- *Pset_FireSuppressionTerminalTypeFireHydrant*
- *Pset_FireSuppressionTerminalTypeSprinkler*
- *Pset_FireSuppressionTerminalTypeHoseReel*

Figure 9-7 to Figure 9-10 show the details regarding the above four property sets based on the published versions given in the IFC 2x2 documentation (Adachi et al., 2003). According to the IFC 2x2 documentation, the *SPRINKLERDEFLECTOR* value also has an associated property set though this could not be identified in the current release.

PropertySet Name	*Pset_FireSuppressionTerminalTypeBreechingInlet*
Applicability	*IfcFireSuppressionTerminalType* entity.
Applicable Classes	*IfcFireSuppressionTerminalType*
Applicable Type Value	*BreechingInlet*
Definition	Symmetrical pipe fitting that unites two or more inlets into a single pipe (BS6100 330 114 adapted).

Property Definitions:

Name	Property Type	Data Type	Definition
BreechingInlet Type	*IfcPropertyEnumerated Value*	*PEnum_BreechingInlet Type* • TWOWAY • FOURWAY • OTHER • USERDEFINED • NOTDEFINED	Defines the type of breeching inlet.
InletDiameter	*IfcPropertySingleValue*	*IfcPositiveLength Measure* / LENGTH UNIT	The inlet diameter of the breeching inlet.
OutletDiameter	*IfcPropertySingleValue*	*IfcPositiveLength Measure* / LENGTH UNIT	The outlet diameter of the breeching inlet.
CouplingType	*IfcPropertyEnumerated Value*	*PEnum_BreechingInlet CouplingType* • INSTANTANEOUS • OTHER • USERDEFINED • NOTDEFINED	Defines the type coupling on the inlet of the breeching inlet.
HasCaps	*IfcPropertySingleValue*	*IfcBoolean*	Does the inlet connection have protective caps.
Material	*IfcPropertyReference Value*	*IfcMaterial*	Material from which the object is constructed

Figure 9-7. The IFC 2x2 *Pset_FireSuppressionTerminalTypeBreechingInlet* property set.

PropertySet Name	*Pset_FireSuppressionTerminalTypeFireHydrant*
Applicability	*IfcFireSuppressionTerminalType* entity.
Applicable Classes	*IfcFireSuppressionTerminalType*
Applicable Type Value	*FireHydrant*
Definition	Device, fitted to a pipe, through which a temporary supply of water may be provided (BS6100 330 6107)

Property Definitions:

Name	Property Type	Data Type	Definition
FireHydrantType	*IfcPropertyEnumeratedValue*	*PEnum_FireHydrantType* • DryBarrel • WetBarrel • Other • NotKnown • Unset	Defines the range of hydrant types from which the required type can be selected where: DryBarrel = A hydrant that has isolating valves fitted below ground and that may be used where the possibility of water freezing is a consideration. WetBarrel = A hydrant that has isolating valves fitted above ground and that may be used where there is no possibility of water freezing.
PumperConnectionSize	*IfcPropertySingleValue*	*IfcPositiveLengthMeasure* / LENGTHUNIT	The size of a connection to which a fire hose may be connected that is then linked to a pumping unit.
NumberOfHoseConnections	*IfcPropertySingleValue*	*IfcInteger*	The number of hose connections on the hydrant (excluding the pumper connection)
HoseConnectionSize	*IfcPropertySingleValue*	*IfcPositiveLengthMeasure* / LENGTHUNIT	The size of connections to which a hose may be connected (other than that to be linked to a pumping unit).
DischargeFlowRate	*IfcPropertySingleValue*	*IfcVolumetricFlowRateMeasure* / VOLUMETRICFLOWRATEUNIT	The volumetric rate of fluid discharge.
FlowClass	*IfcPropertySingleValue*	*IfcLabel*	Alphanumeric indication of the flow class of a hydrant (may be used in connection with or instead of the *FlowRate* property)
WaterIsPotable	*IfcPropertySingleValue*	*IfcBoolean* • Default Value: False	Indication of whether the water flow from the hydrant is potable (set TRUE) or non potable (set FALSE)
PressureRating	*IfcPropertySingleValue*	*IfcPressureMeasure* / PRESSUREUNIT	Maximum pressure that the hydrant is manufactured to withstand.

BodyColor	*IfcPropertySingleValue*	*IfcText*	Color of the body of the hydrant. Note: Consult local fire regulations for statutory colors that may be required for hydrant bodies in particular circumstances.
CapColor	*IfcPropertySingleValue*	*IfcText*	Color of the caps of the hydrant. Note: Consult local fire regulations for statutory colors that may be required for hydrant caps in particular circumstances.

Figure 9-8. The IFC 2x2 *Pset_FireSuppressionTerminalTypeFireHydrant* property set.

PropertySet Name	*Pset_FireSuppressionTerminalTypeSprinkler*
Applicability	*IfcFireSuppressionTerminalType* entity.
Applicable Classes	*IfcFireSuppressionTerminalType*
Applicable Type Value	Sprinkler
Definition	Device for sprinkling water from a pipe under pressure over an area (BS6100 100 3432)

Property Definitions:

Name	Property Type	Data Type	Definition
SprinklerType	*IfcProperty EnumeratedValue*	*PEnum_SprinklerType* • Ceiling • Concealed • Cut-off • Pendant • RecessedPendant • Sidewall • Upright • Other • NotKnown • Unset	Identifies the predefined types of sprinkler from which the type required may be set.
Activation	*IfcProperty EnumeratedValue*	*PEnum_Sprinkler Activation* • Bulb • FusibleSolder • Other • NotKnown • Unset	Identifies the predefined methods of sprinkler activation from which that required may be set.
Response	*IfcProperty EnumeratedValue*	PEnum_Sprinkler Response • Quick • Standard	Identifies the predefined methods of sprinkler response from which that required may be set.

ActivationTemperature	*IfcPropertySingleValue*	*IfcThermodynamic TemperatureMeasure* / THERMODYNAMIC TEMPERATUREUNIT	The temperature at which the object is designed to activate.
CoverageArea	*IfcPropertySingleValue*	*IfcAreaMeasure* / AREAUNIT	The area that the sprinkler is designed to protect.
HasDeflector	*IfcPropertySingleValue*	*IfcBoolean*	Indication of whether the sprinkler has a deflector (baffle) fitted to diffuse the discharge on activation (= TRUE) or not (= FALSE).
BulbLiquidColor	*IfcProperty EnumeratedValue*	*PEnum_SprinklerBulb LiquidColor* • Orange • Red • Yellow • Green • Blue • Mauve • Other • NotKnown • Unset	The color of the liquid in the bulb for a bulb activated sprinkler. Note that the liquid color varies according to the activation temperature requirement of the sprinkler head. Note also that this property does not need to be asserted for quick response activated sprinklers.
DischargeFlowRate	*IfcPropertySingleValue*	*IfcVolumetricFlowrate Measure* / VOLUMETRIC FLOWRATEUNIT	The volumetric rate of fluid discharge.
ResidualFlowing Pressure	*IfcPropertySingleValue*	*IfcPressureMeasure* / PRESSUREUNIT	The residual flowing pressure in the pipeline at which the discharge flow rate is determined.
DischargeCoefficient	*IfcPropertySingleValue*	*IfcReal*	The coefficient of flow at the sprinkler
MaximumWorking Pressure	*IfcPropertySingleValue*	*IfcPressureMeasure* / PRESSUREUNIT	Maximum pressure that the object is manufactured to withstand.
ConnectionSize	*IfcPropertySingleValue*	*IfcPositiveLength* Measure / LENGTHUNIT	Size of the inlet connection to the sprinkler.
FrameMaterial	*IfcProperty ReferenceValue*	*IfcMaterial*	The material used to construct the frame of the sprinkler.
DeflectorMaterial	*IfcProperty ReferenceValue*	*IfcMaterial*	The material used to construct the deflector plate.

Figure 9-9. The IFC 2x2 *Pset_FireSuppressionTerminalTypeSprinkler* property set.

PropertySet Name	*Pset_FireSuppressionTerminalTypeHoseReel*
Applicability	*IfcFireSuppressionTerminalType* entity.
Applicable Classes	*IfcFireSuppressionTerminalType*
Applicable Type Value	*HoseReel*
Definition	A supporting framework on which a hose may be wound (BS6100 155 8201). Note that the service provided by the hose (water/foam) is determined by the context of the system onto which the hose reel is connected.

Property Definitions:

Name	Property Type	Data Type	Definition
HoseReelType	*IfcPropertyEnumeratedValue*	*PEnum_HoseReelType* • Rack • Reel • Other • NotKnown • Unset	Identifies the predefined types of hose arrangement from which the type required may be set.
HoseReelMountingType	*IfcPropertyEnumeratedValue*	*PEnum_HoseReelMountingType* • Cabinet_Recessed • Cabinet_SemiRecessed • Surface • Other • NotKnown • Unset	Identifies the predefined types of hose reel mounting from which the type required may be set.
InletConnectionSize	*IfcPropertySingleValue*	*IfcPositiveLengthMeasure* / LENGTHUNIT	Size of the inlet connection to the hose reel.
HoseDiameter	*IfcPropertySingleValue*	*IfcPositiveLengthMeasure* / LENGTHUNIT	Notional diameter (bore) of the hose.
HoseLength	*IfcPropertySingleValue*	*IfcPositiveLengthMeasure* / LENGTHUNIT	Notional length of the hose fitted to the hose reel when fully extended.
HoseNozzleType	*IfcPropertyEnumeratedValue*	*PEnum_HoseNozzleType* • Fog • StraightStream • Other • NotKnown • Unset	Identifies the predefined types of nozzle (in terms of spray pattern) fitted to the end of the hose from which the type required may be set.
ClassOfService	*IfcPropertySingleValue*	*IfcLabel*	A classification of usage of the hose reel that may be applied.

ClassificationAuthority	*IfcPropertySingleValue*	*IfcLabel*	The name of the authority that applies the classification of service to the hose reel (e.g. NFPA/FEMA)

Figure 9-10. The IFC 2x2 *Pset_FireSuppressionTerminalTypeHoseReel* property set.

9.6 Fire detection and alarm systems

IFC 2x2 includes several entities in the *IfcBuildingControlsDomain* which are relevant to fire detection and alarm systems through the *IfcDistribution-ControlElementType* entity (Figure 9-11). The *IfcSensorType* entity contains several types of sensor either directly or indirectly applicable to fire detection. The *IfcAlarmType* entity contains several types of alarm activation and alarm indication devices.

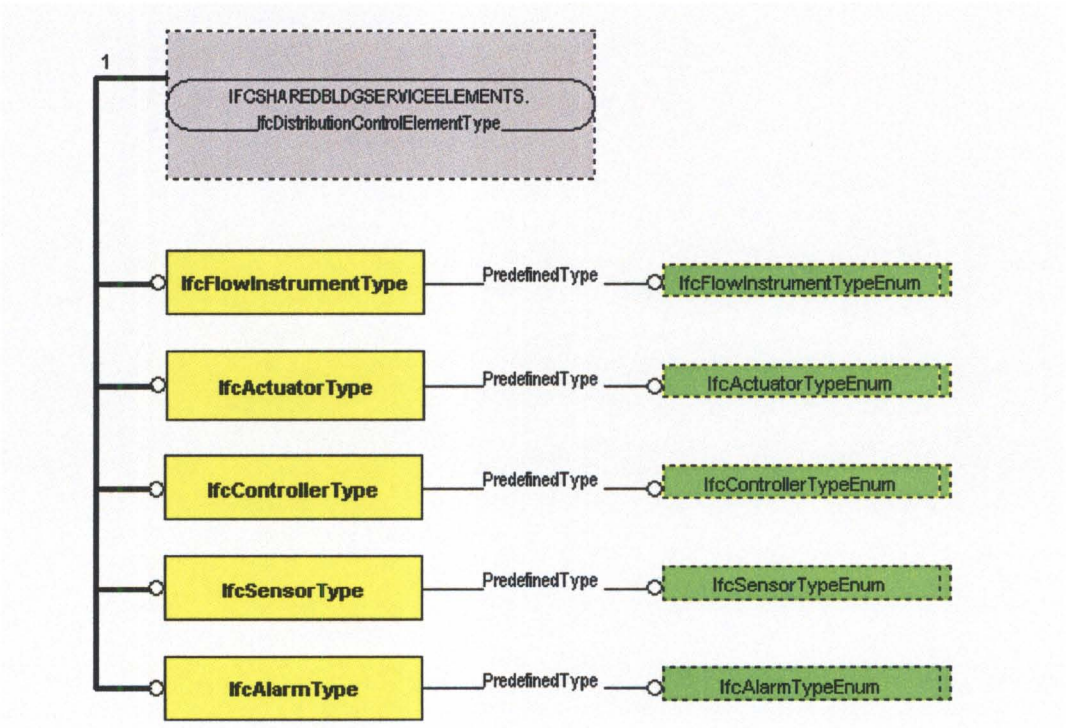


Figure 9-11. *IfcDistributionControlElementType* entity structure.

9.6.1 Fire detection sensors in IFC 2x2

The IFC 2x2 documentation (Adachi et al., 2003) defines the *IfcSensorType* as "a particular type of sensor which is used for detection in a control system". A full enumeration list of the types of sensor defined is given in the *IfcSensorTypeEnum* entity, Figure 9-12 (*USERDEFINED* and *NOTDEFINED* values excluded).

Value	*Definition*
CO2SENSOR	A device that senses or detects carbon dioxide.
FIRESENSOR	A device that senses or detects fire.
FLOWSENSOR	A device that senses or detects flow.
GASSENSOR	A device that senses or detects gas.
HEATSENSOR	A device that senses or detects heat.
HUMIDITYSENSOR	A device that senses or detects humidity.
MOVEMENTSENSOR	A device that senses or detects movement.
PRESSURESENSOR	A device that senses or detects pressure.
SMOKESENSOR	A device that senses or detects smoke.
TEMPERATURESENSOR	A device that senses or detects temperature.

Figure 9-12. IFC 2x2 values and definitions for the *IfcSensorTypeEnum* entity.

In particular, IFC 2x2 defines the fire specific *Pset_SensorTypeFireSensor* property set (Figure 9-13). This entity appears to be a surrogate for a heat detector since the property set includes the *FireSensorSetPoint* property which has a data type of *IfcThermodynamicTemperatureMeasure*. However, this sensor is probably surplus to requirements since *IfcSensorTypeEnum* includes the *Pset_SensorTypeHeatSensor* property set (see below) which fulfils the same role.

PropertySet Name	*Pset_SensorTypeFireSensor*
Applicability	*IfcSensorType* entity.
Applicable Classes	*IfcSensorType*
Applicable Type Value	*FireSensor*
Definition	Definition from IAI: A device that senses or detects the presence of fire.

Property Definitions:

Name	Property Type	Data Type	Definition
FireSensorSetPoint	*IfcPropertySingle Value*	*IfcThermodynamic TemperatureMeasure* / THERMODYNAMIC TEMPERATUREUNIT	The temperature value to be sensed to indicate the presence of fire.
AccuracyOfFireSensor	*IfcPropertySingle Value*	*IfcThermodynamic TemperatureMeasure* / THERMODYNAMIC TEMPERATUREUNIT	The accuracy of the sensor
TimeConstant	*IfcPropertySingle Value*	*IfcTimeMeasure* / TIMEUNIT	The time constant of the sensor.

Figure 9-13. The IFC 2x2 *Pset_SensorTypeFireSensor* property set.

For smoke detection, IFC 2x2 defines the *Pset_SensorTypeSmokeSensor* property set associated with *SMOKESENSOR* (Figure 9-14). It is interesting to note that this property set includes the *IfcBoolean* property *HasBuiltInAlarm* which would allow for a definition of a smoke alarm. The *Pset_SensorTypeSmokeSensor* property set has the potential for expansion since it does not specify the type of smoke sensor and this is discussed further below.

PropertySet Name	*Pset_SensorTypeSmokeSensor*
Applicability	*IfcSensorType* entity.
Applicable Classes	*IfcSensorType*
Applicable Type Value	*SmokeSensor*
Definition	Definition from IAI: A device that senses or detects smoke.

Property Definitions:

Name	Property Type	Data Type	Definition
CoverageArea	*IfcPropertySingle Value*	*IfcAreaMeasure* / AREAUNIT	The floor area that is covered by the sensor (typically measured as a circle whose center is at the location of the sensor)
PressureSensor SetPoint	*IfcPropertySingle Value*	*IfcPositive RatioMeasure*	The smoke concentration value to be sensed.
SmokeSensorRange	*IfcPropertyBounded Value*	*IfcPositive RatioMeasure* • LowerBound: 0 • UpperBound: ?	The upper and lower bounds of smoke concentration for operation of the smoke sensor.
AccuracyOfSmoke Sensor	*IfcPropertySingle Value*	*IfcPositive RatioMeasure*	The accuracy of the sensor
TimeConstant	*IfcPropertySingle Value*	*IfcTimeMeasure* / TIMEUNIT	The time constant of the sensor.
HasBuiltInAlarm	*IfcPropertySingle Value*	*IfcBoolean*	Indicates whether the smoke sensor is included as an element within a smoke alarm/sensor unit (TRUE) or not (FALSE).

Figure 9-14. The IFC 2x2 *Pset_SensorTypeSmokeSensor* property set.

IFC 2x2 also has several other sensor type property sets that are applicable to fire detection systems and these are:

- *Pset_SensorTypeHeatSensor*
- *Pset_SensorTypeTemperatureSensor* (Figure 9-16)
- *Pset_SensorTypeGasSensor*

PropertySet Name	*Pset_SensorTypeHeatSensor*
Applicability	*IfcSensorType* entity.
Applicable Classes	*IfcSensorType*
Applicable Type Value	*HeatSensor*
Definition	Definition from IAI: A device that senses or detects heat.

Property Definitions:

Name	Property Type	Data Type	Definition
CoverageArea	*IfcPropertySingle Value*	*IfcAreaMeasure* / AREAUNIT	The area that is covered by the sensor (typically measured as a circle whose center is at the location of the sensor)
HeatSensor SetPoint	*IfcPropertySingle Value*	*IfcThermodynamic TemperatureMeasure* / THERMODYNAMIC TEMPERATUREUNIT	The temperature value to be sensed.
HeatSensor Range	*IfcPropertyBounded Value*	*IfcThermodynamic TemperatureMeasure* / THERMODYNAMIC TEMPERATUREUNIT • LowerBound: ? • UpperBound: ?	The upper and lower bounds for operation of the temperature sensor.
HeatSensor Accuracy	*IfcPropertySingle Value*	*IfcThermodynamic TemperatureMeasure* / THERMODYNAMIC TEMPERATUREUNIT	The accuracy of the sensor.
TimeConstant	*IfcPropertySingle Value*	*IfcTimeMeasure* / TIMEUNIT	The time constant of the sensor.

Figure 9-15. The IFC 2x2 *Pset_SensorTypeHeatSensor* property set.

PropertySet Name	*Pset_SensorTypeTemperatureSensor*
Applicability	*IfcSensorType* entity.
Applicable Classes	*IfcSensorType*
Applicable Type Value	*TemperatureSensor*
Definition	Definition from IAI: A device that senses or detects temperature.

Property Definitions:

Name	Property Type	Data Type	Definition
TemperatureSensor Type	*IfcProperty EnumeratedValue*	*PEnum_Temperature SensorType* • HighLimit • LowLimit • Outside Temperature • Operating Temperature • RoomTemperature • Other • NotKnown • Unset	Enumeration that Identifies the types of temperature sensor that can be specified.
TemperatureSensor SetPoint	*IfcProperty SingleValue*	*IfcThermodynamic TemperatureMeasure* / THERMODYNAMIC TEMPERATUREUNIT	The temperature value to be sensed.
TemperatureSensor Range	*IfcProperty BoundedValue*	*IfcThermodynamic TemperatureMeasure* / THERMODYNAMIC TEMPERATUREUNIT • LowerBound: ? • UpperBound: ?	The upper and lower bounds for operation of the temperature sensor. May also be termed 'deadband'
AccuracyOfTemperature Sensor	*IfcProperty SingleValue*	*IfcThermodynamic TemperatureMeasure* / THERMODYNAMIC TEMPERATUREUNIT	The accuracy of the sensor
TimeConstant	*IfcProperty SingleValue*	*IfcTimeMeasure* / TIMEUNIT	The time constant of the sensor.

Figure 9-16. The IFC 2x2 *Pset_SensorTypeTemperatureSensor* property set.

It is interesting to note that IFC 2x2 differentiates between temperature sensor and a heat sensor even though they are essentially the same device since they both use temperature as their method of detection. The properties used by these two entities differ slightly in that the *HEATSENSOR* includes a *CoverageArea* property which specifies the area covered by the sensor and the *TEMPERATURESENSOR* includes a *TemperatureSensorType* property which is used to specify the type of sensor. The

existence and nature of the differences suggest that the *HEATSENSOR* is most applicable to fire protection.

9.6.2 Alarm components

The *IfcAlarmType* defines "a device that signals the existence of a condition or situation that is outside the boundaries of normal expectation". The *IfcAlarmTypeEnum* defines the range of different types of alarm that can be specified (Figure 9-17 with *USERDEFINED* and *NOTDEFINED* excluded) and it is noted that the definition mixes alarm activation devices, such as a break glass call point, and alarm indication devices, such as a bell. IFC 2x2 does not provide property sets for any of the *IfcAlarmType* entities.

Value	*Definition*
BELL	An audible alarm
BREAKGLASSBUTTON	An alarm activation mechanism in which a protective glass has to be broken to enable a button to be pressed.
LIGHT	A visual alarm
MANUALPULLBOX	An alarm activation mechanism in which activation is achieved by a pulling action.
SIREN	An audible alarm
WHISTLE	An audible alarm

Figure 9-17. The *IfcAlarmTypeEnum* list of different types of alarm.

9.6.3 Revisions to fire detection related entities in IFC 2x2

Although the fire-related sensor and alarm entities in IFC 2x2 are a useful addition to the product model, they do suffer several limitations. The range of sensors defined needs to be expanded and the terminology could be improved to better meet the conventions used in the fire detection industry.

The range of sensors used in fire detection is currently not adequately covered by the *IfcSensorType*. It is recognised that the *IfcSensorTypeEnum* entity includes the *USERDEFINED* value which can be used to supplement the current defined types but future releases of the IFC Model might include revisions to meet the needs of fire protection.

Although IFC 2x2 includes heat and smoke as two basic types of fire sensor it does not differentiate between the types of heat sensor (e.g. fixed, rate-of-rise) or the type of smoke sensor (photoelectric, ionisation). The generic *GASSENSOR* type would allow for CO sensors though consideration for a specific CO type, similar to the *CO2SENSOR*, might be applicable. Finally, IfcSensorType does not include sensors that use electro-magnetic radiation such as ultra-violet (UV) or infra-red (IR) sensors.

In terms of terminology, IFC 2x2 does not meet the current needs of fire detection technology. Firstly, IFC 2x2 does not easily differentiate between fire detection devices used in commercial-type buildings and those used in the residential environment. Commercial systems have a variety of communication protocols; central control panels; specific wiring and power supply requirements whereas residential systems are usually significantly simpler.

Secondly the relationship between the sensor, alarm indication device and other components are not easily reproduced using the current IFC 2x2 entities. Typically we define a fire detector as one or more sensors and their associated communication electronics (British Standards Institution, 1995). Fire detection devices for residential buildings are typically known as a 'smoke alarm' (British Standards Institution, 1995) or 'single-station smoke alarm' (NFPA, 1996). A smoke alarm consists of: a detector - one or more sensors (e.g. an ionisation chamber) and associated electronics, one or more alerting devices (e.g. the sounder or an escape light) and one or more forms of power supply (e.g. a battery or the mains supply). Although these definitions are specific to the fire detection and alarm industry it is likely that a similar terminology would be appropriate to other industries such as the security alarm business.

Thus the *IfcSensorType* and *IfcAlarmType* entities in IFC 2x2 would need revising to meet the above definitions and consideration should be made to include one or more new entities. One option would be to create a new entity defined as *IfcDetectorType*. The *IfcDetectorType* entity could have an associated *IfcDetectorTypeEnum* which would specify the generic types of detector available such as: *FIREDETECTOR*, *SECURITYDETECTOR* etc. The *IfcDetectorType* entity would then have one or more *IfcSensorType* entities as sub-types. In addition, an *IfcDetectorType* entity could also

have one or more *IfcAlarmType* entities as sub-types in order to handle residential fire detection devices. The *IfcDetectorType* structure could be further expanded to include other sub-types (such as a power supply type entity)

This relationship between the *IfcDetectorType*, *IfcSensorType* and *IfcAlarmType*, entities would still not provide an exact match with the terminology for residential smoke alarms. However, it would be sufficient to capture the essential details of these devices and would integrate with the overall IFC model in a consistent manner.

9.7 Miscellaneous fire protection related properties

9.7.1 Material fire properties

In addition to the material fire properties associated with structural elements (Table 9-3), IFC 2x2 defines material fire properties in several other property sets as shown in Table 9-5.

Property set definition and associated IFC entity	Fire specific property name	Data type	Definition
Pset_CableSegmentType ConductorSegment (IfcCableSegmentType)	*IsFireResistant*	*IfcBoolean*	Indication of whether the sheath is fire resistant (= TRUE) or not (= FALSE).
Pset_CoveringCommon (IfcCovering)	*FireRating*	*IfcLabel*	Fire rating for this object. It is given according to the national fire safety classification.
	FlammabilityRating	*IfcLabel*	Flammability Rating for this object. It is given according to the national building code that governs the rating of flammability for materials.
	Combustible	*IfcBoolean*	Indication whether the object is made from combustible material (TRUE) or not (FALSE).
	SurfaceSpreadOfFlame	*IfcLabel*	Indication on how the flames spread around the surface, It is given according to the national building code that governs the fire behaviour for materials.

Table 9-5. IFC 2x2 material fire properties and associated entities.

The *Pset_CoveringCommon* property set is associated with the *IfcCovering* entity which is used to specify an element which covers some part of another element and is fully dependent on that other element. Coverings are specified with the

IfcCoveringTypeEnum entity and can be one of the following (*USERDEFINED* and *NOTDEFINED* excluded):

- *CEILING* - the covering is used to represent a ceiling
- *FLOORING* - the covering is used to represent a flooring
- *CLADDING* - the covering is used to represent a cladding
- *ROOFING* - the covering is used to represent a roof
- *INSULATION* - the covering is used to insulate an element for thermal or acoustic purposes.
- *MEMBRANE* - an impervious layer that could be used for e.g. roof covering (below tiling - that may be known as sarking *etc.*) or as a damp proof course membrane
- *SLEEVING* - the covering is used to isolate a distribution element from a space in which it is contained.
- *WRAPPING.*

The *IfcDiscreteElement* entity and its associated *Pset_Insulation* property set found in IFC 2x appear to be no longer implemented in IFC 2x2. The *Pset_Insulation* property set included a *FlamabilityRating* [sic] property which is now part of the *Pset_CoveringCommon property* set shown in Table 9-5.

IFC 2x2 also defines an *IfcProductsOfCombustionProperties* entity which is an abstract super-type of the *IfcMaterialProperties* entity. The entity is described as a "common definition to capture the properties of products of combustion generated by elements typically used within the context of building services and flow distribution systems". The following attributes are defined for the entity:

- *SpecificHeatCapacity* - Specific heat of the products of combustion: heat energy absorbed per temperature unit. Usually measured in [J/kg K].
- *N20Content* - Nitrous Oxide (N2O) content of the products of combustion.
- *COContent* - Carbon monoxide (CO) content of the products of combustion.
- *CO2Content* - Carbon Dioxide (CO2) content of the products of combustion.

Apart from *SpecificHeatCapacity*, each attribute is measured in weight of combustion product per unit weight of material.

9.7.2 Risk and reliability

IFC 2x2 has a general risk property set and a general reliability property set defined in the *IfcSharedFacilitiesElements* interoperability layer in addition to the more specific *FireRiskFactor* and *FireHazardFactor* properties given to the *IfcSpace* and *IfcZone* entities (Section 9.2). The two general property sets are associated with high level IFC entities (*i.e. IfcProduct* and *IfcObject*) and have a potential impact on fire engineering.

Property set definition and associated IFC entity	Fire specific property name	Data type	Definition
Pset_Reliability (*IfcProduct*)	*MeanTime BetweenFailure*	*IfcTimeMeasure* / TIMEUNIT	The average time duration between instances of failure of a product.
Pset_Risk (*IfcObject*)	*RiskType*	*PEnum_RiskType* • Business • Hazard • HealthAnd Safety • Insurance • Other • NotKnown • Unset	Identifies the predefined types of risk from which the type required may be set.
	NatureOfRisk	*IfcLabel*	An indication of the nature of the risk that might be encountered. NOTE: It is anticipated that there will be a local agreement that constrains the values that might be assigned to this property.
	Assessment OfRisk	*IfcLabel*	A value that is assigned to the risk that is assessed in a given situation. NOTE: It is anticipated that there will be a local agreement that constrains the values and meanings that may be assigned to this property. The data type is enabled as an IfcLabel to account for alpha, numerical and alphanumerical assessment ratings. Note also that assessment of risk may frequently be associated with the physical location of the object for which the risk is assessed.

Table 9-6. IFC 2x2 risk and reliability property sets.

9.8 Conclusions

This review shows that there are many fire protection engineering related entities and properties that exist in IFC 2x2 but they are scattered across domains and could benefit from some revision. The fact that IFC 2x2 includes the *IfcPlumbingFire-*

ProtectionDomain shows that there is an appreciation of fire protection needs but this is not limited to extinguishing systems and suggests that fire protection could become its own separate domain in any future revision to the IFC model. However, it is also recognised that many fire protection components could just as logically incorporated into a number of other domains, as is the case in IFC 2x2, making the justification for a separate fire protection domain more difficult to make.

Chapter 10:
FIRE ENGINEERING PROPERTIES IN THE IFC BUILDING PRODUCT MODEL AND MAPPING TO BRANZFIRE

Spearpoint M J, Fire engineering properties in the IFC building product model and mapping to BRANZFIRE. Accepted for publication in the International Journal on Engineering Performance-Based Fire Codes, June 2005.

Paper Referees:
Two anonymous reviewers

10.1 Abstract

The Industry Foundation Class (or IFC) Model is a standardised, object-oriented 'building product model' that provides an electronic description of buildings. Entities are defined in the model to represent building elements with their associated properties. This paper reviews the latest release of the IFC Model considering entities and properties related to fire engineering. The paper goes on to examine how these entities and properties can be mapped to the input requirements of the **BRANZFIRE** fire simulation software.

10.2 Introduction

Performance-based fire engineering design often involves the development and analysis of fire scenarios using fire simulation software (commonly referred to as 'fire models'). In order to carry out the design process, a considerable amount of time can be spent specifying the input to the simulation software prior to execution. Information such as the geometry and topology of a building, the location of items within the building and the properties of those items may be required. In addition the fire performance measures associated with the structure, the building contents and fire protection equipment are often desired as inputs to fire simulation software. All of this information can be quite voluminous and needs to be accurately transferred to the simulation software. Mistakes and missing information can lead to inappropriate output.

Traditionally, such information is entered manually even when the building plans may have been generated by Computer Aided Design (CAD) software. Yet where there is the ability to directly transfer CAD data into fire simulation software there can be significant limitations. These limitations stem from a variety of reasons including where the fire simulation software tool may be incapable of interpreting any kind of CAD file or where the CAD file does not adequately describe the building in a form from which the software can extract relevant details. Various solutions to the transfer of CAD data into fire simulation software and more general simulation software have been addressed elsewhere. Some researchers have investigated techniques to automate the recognition of paper-based drawings in a form that can be interpreted by CAD software (Berkhahn and Esch, 2003). Massa and Cappuccio (1995) investigated the feasibility of adapting a proprietary topological analysis CAD system for modelling bomb blast effects as an interface to fire simulation software and concluded that implementation was readily

achievable although it is unclear whether any further development was undertaken. Frost, Patel, Galea, Rymacrzyk and Mawhinney (2001) developed methods to transfer DXF files into the **SMARTFIRE** computational fluid dynamics fire simulation software. Similarly, the **Simulex** evacuation model (Thompson and Marchant, 1995) allows the direct import of CAD data through DXF files. Often these methods require that the original files be manually "cleaned" of data that would otherwise be incorrectly interpreted by the target software. The limitations of the DXF format also mean that only basic geometry primitives can be derived from the files because the format does not allow more detailed definition of objects (Anon, 2003).

Overcoming the current limitations of data exchange between CAD and fire simulation software requires a much richer description of buildings than typical formats such as DXF can provide. Mowrer and Williamson (1988) identified the key features required by CAD systems to permit integration with fire simulation software. These characteristics included object-orientation, the association of attributes with objects and the ability to extract attributes from a CAD-developed drawing database. The IFC Model implements many of these features and thus is ideally suited to further investigation.

10.3 Fire simulation software

A recent survey (Olenick and Carpenter, 2003) has shown that there are a wide range of fire simulation software tools available. The tools can be used for various fire engineering related tasks such as predicting fire and smoke spread, determining the performance of structural elements under fire conditions and the analysis of people movement in buildings or other premises. The software tools vary in the extent of the fire hazard scenario represented and subsequently the complexity of the input requirements and the sophistication of the output capabilities.

Zone models are a common category of fire simulation software available to the fire engineer. Quintiere (2002) provides a description of the basic conservation equations and relationships that are used by most zone fire simulation software. The atmosphere within a compartment is normally split into two zones; the hot upper gas layer due to the fire and the cool layer below. The physical conditions within these layers are considered vertically and horizontally uniform. The fire plume transports combustion

products from the lower layer to the upper layer and gases flow through vents in compartment boundaries. Although zone fire simulation tools all follow the same basic philosophy regarding the way in which the fire environment is represented, individual software tools may have facilities that are not present in others.

BRANZFIRE (Wade 2003a, Wade 2003b) is a widely available multi-compartment zone model. It can simulate the movement of smoke between up to 10 inter-connected spaces. Fires are specified by a rate of heat release curve or using a built-in fire spread model in the case of room linings. The model also has the ability to incorporate sprinkler and smoke detector activation, the breaking of window glass (Parry, Wade and Spearpoint, 2003) and the effects of mechanical fans. Although the mapping of the IFC Model to the **BRANZFIRE** fire simulation software is specifically explored in this paper, the issues are representative of those faced integrating many of the available zone fire simulation software family.

10.4 Product models

10.4.1 General description

In general, any product can be considered to consist of a collection of 'entities'. A 'product model' expresses the type of entities that represent the product; the properties that are needed to describe those entities and the inter-relationship between entities. The description of a product model is commonly known as its 'schema'. A 'building product model' is a product model that specifically relates to buildings where entities may be physical objects such as doors, windows, walls etc. or more conceptual entities such as spaces or processes, contractual details etc.

Fire engineering is one of many domains including architecture, structural engineering, environmental engineering and building services which can benefit from using building product models already noted in Chapter 4. Many parameters related to a building are common to a range of disciplines including fire engineering. These parameters may include the building geometry and topology, the materials and components used in the construction and the location of the structure within the broad environment. However, due to the specialised needs of fire engineering, there are also parameters that are unique to the domain.

10.4.2 Product model scope

Product models can be thought of being of two types; either they are general or they are domain specific (Ito, 1995). A general product model supports the generation and sharing of project data through the complete building lifecycle amongst a diverse range of domains. A general product model does not attempt to include every aspect of a product as this would likely be too complex and take too long to develop. Instead a general product model describes entity types at relatively high level.

Conversely, domain specific models retain as much of the project data as required for use within a domain. For the fire engineering domain it is likely that a general product model will not provide all of the detail needed for fire simulation software tools or any other tasks related to fire engineering. A domain specific product model could be created which has all those entities relevant to the fire engineering domain but this can lead to problems when sharing that project data with participants in other domains. There would need to be a number of software tools available to interpret each domain specific model. Furthermore, domains outside fire engineering might find the domain specific model has essential information missing or not in a form that is useable by them.

Even where a product model completely describes a domain, it is possible that specific software tools may only implement a reduced proportion of the whole product model. This can mean that although the product model has an entity described within its schema, the software tool cannot be used to create a specific instance of that entity or completely populate the properties associated with the entity. In some cases it may be possible to manually add entities in lieu of having an appropriate software tool.

Finally the interpretation of the product model may lead to complications. The structure of entities may not be compatible with the specific requirements of the target software tool. This can happen where there is insufficient detail in the product model but also where the requirements of the software tool include simplifications and assumptions about a product that need to be accounted for during the data transfer process.

10.4.3 IFC Model

The IFC Model is a general building product model that began development around 1996 and has gone through several major releases to date. The latest version of the IFC

Model is 2x Edition 2 (Liebich, 2003) referred to as 'IFC 2x2' in this paper, with corrections published in Addendum 1. Primarily IFC files are exchanged using STEP (Standard for the Exchange of Product model data) technology, ISO 10303 (ISO, 2002).

The IFC Model addresses the limited scope of a general product model using 'property set definitions'. The high level entities terminate at 'leaf nodes' which allow object types to be extended using the property set definition sub-schema. The specification of property set definitions can be made outside of the main IFC Model by specialists within their domain. Even with the use of property set definitions, there may still be specific object types missing or incomplete simply due to the fact that nobody has yet included the information in the product model. This has been the case with many of the entities that might be of use to fire engineers. A review of the properties in the earlier version 2x of the IFC Model with respect to fire engineering found that whilst there are a number of fundamental material properties and several regulatory-related properties in the IFC Model, there are also many areas in which the model can be extended (as described in Chapter 7). Release IFC 2x2 has significantly greater support for fire engineering than the previous IFC 2x Model. In particular, IFC 2x2 has a specific domain referred to as the *IfcPlumbingFireProtectionDomain* which covers aspects of fire extinguishing systems. However, there are many other entities and properties that are relevant to fire engineering scattered throughout the IFC 2x2 Model and these are mentioned briefly in this chapter. Figure 10-1, adapted from Liebich (2004), shows the structure of the IFC Model and the underlined text indicates items that are applicable to this chapter.

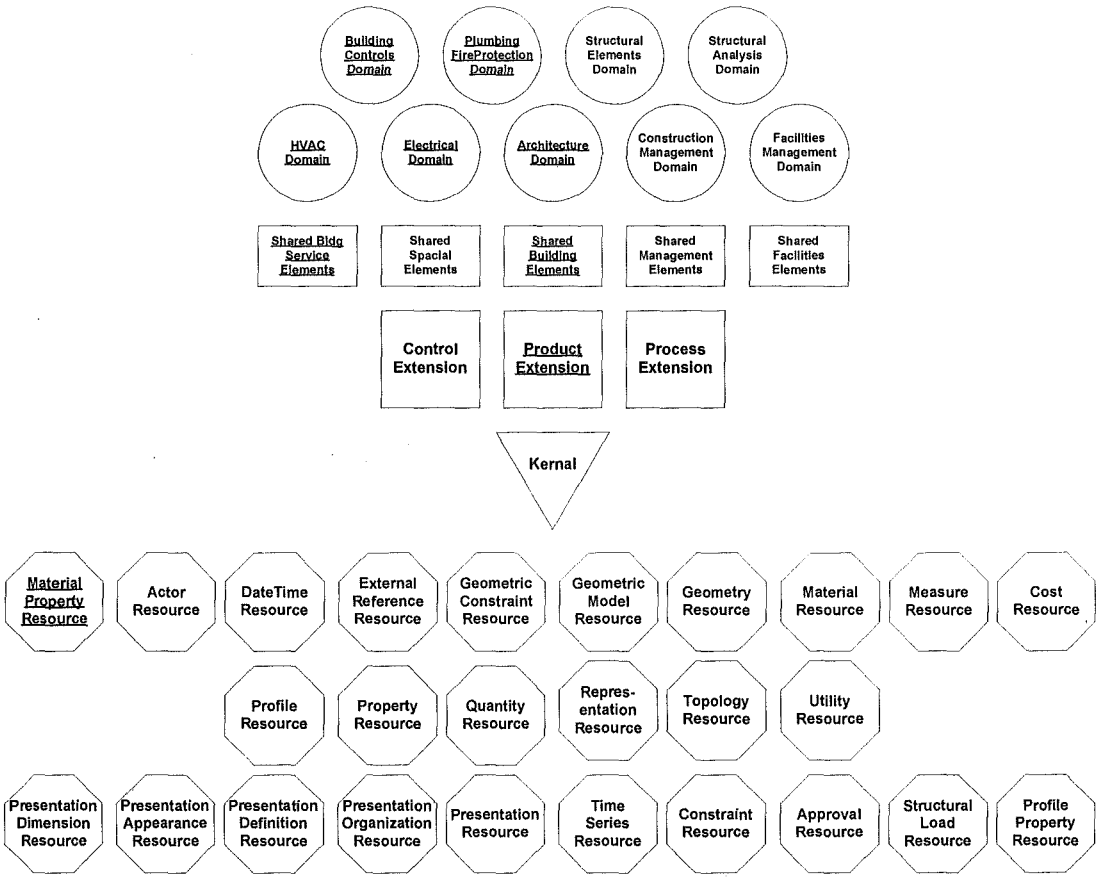


Figure 10-1. IFC 2x2 schema overview.

The contents of the IFC Model are identified with a specific convention which is followed in this paper and is indicated hereafter in italicised text. Entities are prefixed with *Ifc*; property sets prefixed with *Pset* and enumerated lists of entities or properties with the addition of *Enum.*

10.5 Property set mappings

10.5.1 Building spaces

The IFC Model has entities that relate to buildings as a whole and the spaces that form those buildings. Several of these entities have properties that relate directly to fire engineering. Zones and spaces have specific fire engineering properties defined in the *Pset_SpaceFireSafetyRequirements* property set. The properties consider the use of the space through the *MainFireUse*, *AncillaryFireUse* and *FlammableStorage* properties; associated risks and hazards with the *FireRiskFactor* and *FireHazardFactor* properties; provisions for means of escape with the *FireExit* property and any active fire protection systems in the space by the use of the *SprinklerProtection*, *SprinklerProtection-*

Automatic and *AirPressurization* Boolean properties. Similar properties are associated with the *IfcBuilding* and *IfcBuildingStorey* entities with a supplementary *ProtectedOpening* property associated with the *Pset_OpeningElement* property set.

In addition to the means of escape provisions specified in the property sets associated with buildings and spaces, IFC 2x2 also includes properties that provide the occupancy characteristics of spaces in the *IfcArchitectureDomain* layer. The *OccupancyType* is defined according to the presiding national building code and the *OccupancyNumber* specifies the maximum number of occupants for the designed usage of the space.

10.5.2 Structural elements

Table 10-1 shows the fire engineering properties for structural elements defined in IFC 2x2. The properties identify fire and smoke containment with the *FireRating*, *SmokeStop*, *SelfClosing* and *Compartmentation* properties; the performance of materials with the *Combustible* and *SurfaceSpreadOfFlame* properties and means of escape provisions through the *FireExit* property (similar to the property associated with buildings and spaces).

Property name and data type	Definition	Property set definition and associated IFC entity									
		Pset_BeamCommon (IfcBeam)	*Pset_ColumnCommon (IfcColumn)*	*Pset_CurtainWallCommon (IfcCurtainWall)*	*Pset_DoorCommon (IfcDoor)*	*Pset_RampCommon (IfcRamp)*	*Pset_RoofCommon (IfcRoof)*	*Pset_SlabCommon (IfcSlab)*	*Pset_StairCommon (IfcStair)*	*Pset_WallCommon (IfcWall, IfcWallStandardCase)*	*Pset_WindowCommon (IfcWindow)*
FireRating (IfcLabel)	Fire rating for object as defined by a national fire safety classification.	X	X	X	X	X	X	X	X	X	X
Combustible (IfcBoolean)	Indicates whether object is made from combustible material.		X					X		X	
SurfaceSpread OfFlame (IfcLabel)	Surface flame spread as defined by a national building code that governs fire behaviour of materials.		X					X		X	
FireExit (IfcBoolean)	Indicates whether object is designed to serve as an exit in the case of fire as defined by a national building code.				X	X			X		
SelfClosing (IfcBoolean)	Indicates whether object is designed to close automatically after use.				X						
SmokeStop (IfcBoolean)	Indicates whether object is designed to provide a smoke stop.				X						X
Compartmentation (IfcBoolean)	Indicates whether object is designed to serve as fire compartmentation.							X		X	

Table 10-1. Fire engineering properties for structural elements defined in IFC 2x2.

IFC 2x2 also includes a general property set *Pset_FireRatingProperties* in the *IfcSharedBldgServiceElements* interoperability layer. This property set includes a

FireResistanceRating property which appears to fulfil the same function as the *FireRating* property used for the fire resistance of structural elements. Similarly, an *IsCombustible* property in *Pset_FireRatingProperties* appears to fulfil the same function as the *Combustible* property for structural elements.

The fire engineering properties related to building spaces and structural elements have potential contribution to the regulatory environment and as input to other fire simulation software but in terms of mapping to **BRANZFIRE**, these properties have no directly equivalent parameters.

10.5.3 Fire suppression systems

The *IfcPlumbingFireProtectionDomain* contains several property sets related to fire extinguishing systems. The IFC 2x2 documentation (Adachi et al., 2003) defines the *IfcFireSuppressionTerminalType* as a "particular type of *IfcFlowTerminal* that has the purpose of delivering a fluid (gas or liquid) that will suppress a fire" and includes all forms of sprinkler, spreader and other form of terminal that is connected to a pipe system. The specific types of *IfcFireSuppressionTerminalType* are defined in Table 10-2, adapted from Adachi et al., 2003.

Value	Definition
BREECHINGINLET	Symmetrical pipe fitting that unites two or more inlets into a single pipe.
FIREHYDRANT	Device, fitted to a pipe, through which a temporary supply of water may be provided.
HOSEREEL	A supporting framework on which a hose may be wound.
SPRINKLER	Device for sprinkling water from a pipe under pressure over an area.
SPRINKLERDEFLECTOR	Device attached to a sprinkler to deflect the water flow into a spread pattern to cover the required area.

Table 10-2. IFC 2x2 values and definitions for the *IfcFireSuppressionTerminal-TypeEnum* entity.

The four property sets associated with the *IfcFireSuppressionTerminalType* are *Pset_FireSuppressionTerminalTypeBreechingInlet*, *Pset_FireSuppressionTerminal-TypeFireHydrant*, *Pset_Fire-SuppressionTerminalTypeHoseReel and Pset_Fire-SuppressionTerminalTypeSprinkler*. The first three of these property sets are not applicable to **BRANZFIRE** because there are no directly equivalent parameters

however the *Pset_FireSuppressionTerminalTypeSprinkler* is of particular relevance since **BRANZFIRE** can be used to predict sprinkler activation and sprinkler suppression of fires.

Figure 10-2, adapted from Adachi et al., 2003, shows the details regarding the *Pset_FireSuppressionTerminal-TypeSprinkler* property set. **BRANZFIRE** requires the Response Time Index (RTI) and Conduction (C) factor, activation temperature, radial distance to fire and distance below ceiling. Of these, the last parameter can be obtained from the space geometry and the radial distance may be determined if the location of the fire can be identified but only the *ActivationTemperature* is explicitly supplied in the property set. **BRANZFIRE** has the option to specify the water spray density and this can be obtained from *Pset_Fire-SuppressionTerminalTypeSprinkler* through the *CoverageArea* and *DischargeFlowRate* properties.

The addition of properties for simulation purposes such as an RTI and C-factor and extending the range of characteristics available in the *SprinklerType, Activation* and *Response* properties would be beneficial as further enhancement to the *Pset_Fire-SuppressionTerminalTypeSprinkler* property set in future releases of the IFC Model.

PropertySet Name	*Pset_FireSuppressionTerminalTypeSprinkler*
Applicability	*IfcFireSuppressionTerminalType* entity.
Applicable Type Value	*SPRINKLER*
Definition	Device for sprinkling water from a pipe under pressure over an area.

Property Definitions:

Name	Data Type	Definition
SprinklerType	*PEnum_SprinklerType*† • Ceiling • Concealed • Cut-off • Pendant • RecessedPendant • Sidewall • Upright	Identifies the predefined types of sprinkler from which the type required may be set.
Activation	*PEnum_Sprinkler Activation*† • Bulb • FusibleSolder	Identifies the predefined methods of sprinkler activation from which that required may be set.

Response	*PEnum_Sprinkler Response* • Quick • Standard	Identifies the predefined methods of sprinkler response from which that required may be set.
ActivationTemperature	*IfcThermodynamic TemperatureMeasure*	The temperature at which the object is designed to activate.
CoverageArea	*IfcAreaMeasure*	The area that the sprinkler is designed to protect.
HasDeflector	*IfcBoolean*	Indication of whether the sprinkler has a deflector (baffle) fitted to diffuse the discharge on activation.
BulbLiquidColor	*PEnum_SprinklerBulb LiquidColor*† • Orange • Red • Yellow • Green • Blue • Mauve	The colour of the liquid in the bulb for a bulb activated sprinkler. Note that the liquid colour varies according to the activation temperature requirement of the sprinkler head. Note also that this property does not need to be asserted for quick response activated sprinklers.
DischargeFlowRate	*IfcVolumetricFlowrate Measure*	The volumetric rate of fluid discharge.
ResidualFlowingPressure	*IfcPressureMeasure*	The residual flowing pressure in the pipeline at which the discharge flow rate is determined.
DischargeCoefficient	*IfcReal*	The coefficient of flow at the sprinkler
MaximumWorkingPressure	*IfcPressureMeasure*	Max. pressure object is manufactured to withstand.
ConnectionSize	*IfcPositiveLength Measure*	Size of the inlet connection to the sprinkler.
FrameMaterial	*IfcMaterial*	Material used to construct the frame of sprinkler.
DeflectorMaterial	*IfcMaterial*	The material used to construct the deflector plate.

† also includes Other, NotKnown and Unset values.

Figure 10-2. The IFC 2x2 *Pset_FireSuppressionTerminalTypeSprinkler* property set.

10.5.4 Fire detection and alarm systems

IFC 2x2 includes several entities in the *IfcBuildingControlsDomain* which are relevant to fire detection and alarm systems through the *IfcDistributionControlElementType* entity. The *IfcSensorType* entity contains several types of sensor either directly or indirectly applicable to fire detection. The *IfcAlarmType* entity contains several types of alarm activation and alarm indication devices. The *IfcAlarmType* defines "a device that signals the existence of a condition or situation that is outside the boundaries of normal expectation", and defines the range of different types of alarm that can be specified. However IFC 2x2 does not provide property sets for any of the *IfcAlarmType* entities. Since **BRANZFIRE** does

not include any human behaviour modelling, these properties are not relevant to the current mapping exercise.

The IFC 2x2 documentation (Adachi et al., 2003) defines the *IfcSensorType* as "a particular type of sensor which is used for detection in a control system". A list of the defined types of sensor is given in the *IfcSensorTypeEnum* enumeration type as shown in Table 10-3, adapted from Adachi et al., 2003.

Value	Definition
CO2SENSOR	A device that senses or detects carbon dioxide.
FIRESENSOR	A device that senses or detects fire.
FLOWSENSOR	A device that senses or detects flow.
GASSENSOR	A device that senses or detects gas.
HEATSENSOR	A device that senses or detects heat.
HUMIDITYSENSOR	A device that senses or detects humidity.
MOVEMENTSENSOR	A device that senses or detects movement.
PRESSURESENSOR	A device that senses or detects pressure.
SMOKESENSOR	A device that senses or detects smoke.
TEMPERATURESENSOR	A device that senses or detects temperature.

Table 10-3. IFC 2x2 values and definitions for the *IfcSensorTypeEnum* entity.

IFC 2x2 has several thermal sensor type property sets that are potentially applicable to fire detection systems and could be mapped to **BRANZFIRE**. IFC 2x2 differentiates between a temperature sensor and a heat sensor even though they are essentially the same device since they both use temperature as their method of detection. The properties used by these two entities differ slightly in that the *HEATSENSOR* includes a *CoverageArea* property which specifies the area covered by the sensor (Figure 10-3, adapted from Adachi et al., 2003) and the *TEMPERATURESENSOR* includes a *TemperatureSensorType* property which is used to specify the type of sensor. The existence and nature of the differences suggest that the *HEATSENSOR* is most applicable to fire protection.

PropertySet Name	*Pset_SensorTypeHeatSensor*
Applicability	*IfcSensorType* entity.
Applicable Type Value	*HEATSENSOR*
Definition	A device that senses or detects heat.

Property Definitions:

Name	Data Type	Definition
CoverageArea	*IfcAreaMeasure*	The area that is covered by the sensor (typically measured as a circle whose centre is at the location of the sensor)
HeatSensorSetPoint	*IfcThermodynamic TemperatureMeasure*	The temperature value to be sensed.
HeatSensorRange	*IfcThermodynamic TemperatureMeasure* • LowerBound: variable • UpperBound: variable	The upper and lower bounds for operation of the temperature sensor.
HeatSensorAccuracy	*IfcThermodynamic TemperatureMeasure*	The accuracy of the sensor.
TimeConstant	*IfcTimeMeasure*	The time constant of the sensor.

Figure 10-3. The IFC 2x2 *Pset_SensorTypeHeatSensor* property set.

IFC 2x2 also defines the a *Pset_SensorTypeFireSensor* property set (Figure 10-4, adapted from Adachi et al., 2003) which appears to be a surrogate for a heat detector since the property set includes the *FireSensorSetPoint* property which has a data type of *IfcThermodynamic-TemperatureMeasure*. However, this sensor appears to be surplus to requirements with the inclusion of the *Pset_SensorTypeHeatSensor* property set which essentially fulfils the same role.

PropertySet Name	*Pset_SensorTypeFireSensor*
Applicability	*IfcSensorType* entity.
Applicable Type Value	*FIRESENSOR*
Definition	A device that senses or detects the presence of fire.

Property Definitions:

Name	Data Type	Definition
FireSensorSetPoint	*IfcThermodynamic TemperatureMeasure*	The temperature value to be sensed to indicate the presence of fire.
AccuracyOfFireSensor	*IfcThermodynamic TemperatureMeasure*	The accuracy of the sensor
TimeConstant	*IfcTimeMeasure*	The time constant of the sensor.

Figure 10-4. The IFC 2x2 *Pset_SensorTypeFireSensor* property set.

For smoke detection, IFC 2x2 defines the *Pset_SensorTypeSmokeSensor* property set associated with *SMOKESENSOR* (Figure 10-5, adapted from Adachi et al., 2003). It is interesting to note that this property set includes the *HasBuiltInAlarm* property which would allow for a definition of a smoke alarm.

BRANZFIRE requires the optical density characteristics that will activate the alarm which can be defined as a specified optical density or using built-in sensitivity levels appropriate to AS1603.2 (Standards Australia, 1989). **BRANZFIRE** also accounts for the time delay for smoke entry into the detection chamber. The *Pset_SensorTypeSmokeSensor* property set includes the (apparently inaccurately named) *PressureSensorSetPoint* property which could be used to define the activation optical density and the *TimeConstant* property which could be used to specify the entry delay. The *Pset_SensorTypeSmokeSensor* property set has the potential for expansion since it does not specify the type of smoke sensor although this is not specifically required by **BRANZFIRE**.

PropertySet Name	*Pset_SensorTypeSmokeSensor*
Applicability	*IfcSensorType* entity.
Applicable Type Value	*SMOKESENSOR*
Definition	A device that senses or detects smoke.

Property Definitions:

Name	Data Type	Definition
CoverageArea	*IfcAreaMeasure*	The floor area that is covered by the sensor (typically measured as a circle whose centre is at the location of the sensor)
PressureSensorSetPoint	*IfcPositiveRatioMeasure*	The smoke concentration value to be sensed.
SmokeSensorRange	*IfcPositiveRatioMeasure* • LowerBound: 0 • UpperBound: variable	The upper and lower bounds of smoke concentration for operation of the smoke sensor.
AccuracyOfSmokeSensor	*IfcPositiveRatioMeasure*	The accuracy of the sensor
TimeConstant	*IfcTimeMeasure*	The time constant of the sensor.
HasBuiltInAlarm	*IfcBoolean*	Indicates whether the smoke sensor is included as an element within a smoke alarm/sensor unit.

Figure 10-5. The IFC 2x2 *Pset_SensorTypeSmokeSensor* property set.

IFC 2x2 also provides the *Pset_SensorTypeGasSensor* property set (Figure 10-6, adapted from Adachi et al., 2003) which would allow for the inclusion of fire detection by CO or other suitable combustion product through the *GasDetected* property. The property set is similar to those for other forms of fire detection but does not include a *CoverageArea* property. Since the current version of **BRANZFIRE** does not allow for detection by combustion gases, mappings are not applicable.

PropertySet Name	*Pset_SensorTypeGasSensor*
Applicability	*IfcSensorType* entity.
Applicable Type Value	*GASSENSOR*
Definition	A device that senses or detects gas.

Property Definitions:

Name	Data Type	Definition
GasDetected	*IfcLabel*	Identification of the gas that is being detected.
GasSensorSetPoint	*IfcPositiveRatioMeasure*	The gas concentration value to be sensed.
GasSensorRange	*IfcPositiveRatioMeasure* • LowerBound: 0 • UpperBound: variable	The upper and lower bounds of gas concentration for operation of the gas sensor.
AccuracyOfGasSensor	*IfcPositiveRatioMeasure*	The accuracy of the sensor
TimeConstant	*IfcTimeMeasure*	The time constant of the sensor.

Figure 10-6. The IFC 2x2 *Pset_SensorTypeGasSensor* property set.

10.5.5 Smoke control

IFC 2x2 includes a wide range of property sets applicable to HVAC systems through the *IfcHvacDomain* layer. These property sets include those appropriate to fans, ducts, heating and cooling systems and many other components. For fans the *Pset_FanTypeCommon* includes a number of general properties that are not specifically applicable to smoke control and **BRANZFIRE** mapping. However IFC 2x2 includes a property set for smoke control systems in the *Pset_FanTypeSmokeControl* property set (Figure 10-7, adapted from Adachi et al., 2003).

Additional properties for fans are also available through the *Pset_FlowMoving-DeviceFan* property set and specifically the *ApplicationOfFan* property can be used to indicate whether the fan type is *SUPPLY*, *EXHAUST* or has other applications.

PropertySet Name	*Pset_FanTypeSmokeControl*
Applicability	*IfcFanType*
Definition	Smoke control attributes of fan participating as part of a smoke control system.

Property Definitions:

Name	Data Type	Definition
OperationalCriteria	*IfcTimeMeasure*	Time of operation at maximum operational ambient air temperature.
MaximumDesignTemperature	*IfcThermodynamic TemperatureMeasure*	Maximum design operational temperature.
SmokeControlFlowrate	*IfcVolumetricFlowRate Measure*	Flowrate of fan while operating as a part of the smoke control system.

Figure 10-7. The *Pset_FanTypeSmokeControl* property set.

BRANZFIRE allows a single fan to be associated with each room. The **BRANZFIRE** input parameters for mechanical ventilation are the flow rate, fan start time, cross-fan pressure limit, fan elevation, whether the fan is extracting or pressurising the space and if a fan curve should be used. The flow rate and air flow direction can be derived from the *SmokeControlFlowrate* and *ApplicationOfFan* properties respectively. The elevation of the fan can be determined through the geometry of the space but other parameters would need to be specified by the **BRANZFIRE** user.

Dampers are also defined in the IFC 2x2 *IfcHvacDomain* and in terms of fire engineering, they contribute to the fire resistance rating and smoke containment of a structure. Dampers are specified using the *IfcDamperType* entity which is a sub-type of the *IfcFlowControllerType* entity. IFC 2x2 defines three types of damper and are described as (Adachi et al., 2003):

- *FIREDAMPER*: Used to prevent the spread of fire for a specified duration.
- *SMOKEDAMPER*: Used to prevent the spread of smoke.
- *FIRESMOKEDAMPER*: Combination fire and smoke damper used to prevent the spread of fire and smoke.

There are no provisions to include dampers in **BRANZFIRE** and the details of the associated property sets are not included here.

10.5.6 Material properties

10.5.6.1 Fire resistance

In addition to the material fire properties associated with structural elements, IFC 2x2 defines material fire properties in two other property sets as shown in Table 10-4. The *Pset_CoveringCommon* property set is associated with the *IfcCovering* entity which is used to specify an element which covers some part of another element and is fully dependent on that other element. Definitions for *Pset_CoveringCommon* follow those given for structural elements.

Property set definition and associated IFC entity	Property name and data type	Definition
Pset_CableSegmentType ConductorSegment (IfcCableSegmentType)	*IsFireResistant (IfcBoolean)*	Indicates whether sheath is fire resistant.
Pset_CoveringCommon (IfcCovering)	*FireRating (IfcLabel)*	Fire rating for this object.
	FlammabilityRating (IfcLabel)	Flammability rating for this object.
	Combustible (IfcBoolean)	Indicates whether object is made from combustible material.
	SurfaceSpreadOfFlame (IfcLabel)	Indication of surface flame spread.

Table 10-4. IFC 2x2 material fire properties and associated entities.

10.5.6.2 Fire growth

For a large majority of fire simulation software tools it is necessary to identify the material properties related to the ignition and burning of building contents and linings. In **BRANZFIRE** burning items are characterised by a rate of heat release curve and the spread of fire over room linings can be determined from material properties. For fires involving linings, **BRANZFIRE** requires the material properties of the lining be specified plus an associated cone calorimeter data file (Wade, 2003a). Material properties held by the **BRANZFIRE** thermal properties database are the material description, thermal conductivity, specific heat, density, emissivity, minimum temperature for spread, flame spread parameter, heat of combustion, soot yield, water yield and carbon dioxide yield.

IFC 2x2 defines a number of material property entities which are sub-types of an *IfcMaterialProperties* entity. These sub-types contain related collections of properties

that includes mechanical, thermal, optical and several others. For mapping to the BRANZFIRE thermal database, density is provided in *IfcGeneralMaterialProperties* entity; and heat capacity and thermal conductivity in *IfcThermalMaterialProperties* entity. IFC 2x2 also defines an *IfcProductsOfCombustionProperties* sub-type with the following attributes defined for the entity:

- *SpecificHeatCapacity* - Specific heat of the products of combustion, J/kg K.
- *N20Content* - Nitrous oxide content of the products of combustion.
- *COContent* - Carbon monoxide content of the products of combustion.
- *CO2Content* - Carbon dioxide content of the products of combustion.

Apart from *SpecificHeatCapacity*, each attribute is measured in weight of combustion product per unit weight of material. Finally, IFC 2x2 has an *IfcFuelProperties* sub-type which includes properties that specify the amount of energy released by a fuel when completely burned, the combustion temperature of the material and the carbon content ratio. The *IfcProductsOfCombustionProperties* and *IfcFuelProperties* sub-types are described in the IFC documentation (Adachi et al., 2003) as being those properties "typically used within the context of building services and flow distribution systems". Since they are not intended for a fire engineering domain they have limited applicability for fire simulation software and are not considered relevant to **BRANZFIRE**.

In order to identify rate of heat release characteristics of an item it is clear that the current property specifications of IFC 2x2 cannot be used. Since the rate of heat release is a specialised aspect of the fire engineering domain, it would not be expected to be included in a general product model such as the IFC Model. Instead a transformation that allows the creation of a new property set (provisionally referred to as *Pset_FurnitureHeatRelease*) from the FireBaseXML database schema has been created (Figure 7-2). The addition of a *Pset_FurnitureHeatRelease* property set to an item may also provide a means to identify the location of the fire which is relevant to several geometric inputs required by **BRANZFIRE** such as the radial distance between a sprinkler and a fire.

10.5.6.3 Glazing

BRANZFIRE has the capability to predict when glass breaks (Parry, Wade and Spearpoint, 2003) and this procedure requires a number of properties for the glass to be supplied. Required properties are the glass thickness, thermal conductivity, thermal diffusivity, Young's modulus, fracture stress, shading depth, thermal expansion coefficient and optionally the distance to the flame.

PropertySet Name	*Pset_DoorWindowGlazingType*
Applicability	*IfcDoor*, *IfcWindow* entity.
Definition	Properties common to the definition of the glazing component of occurrences of *IfcDoor* and *IfcWindow*, used for thermal and lighting calculations.

Property Definitions:

Name	Data Type	Definition
GlassLayers	*IfcCountMeasure* • Default Value: 2	Number of glass layers within the frame.
GlassThickness1	*IfcPositiveLengthMeasure*	Thickness of the first (inner) glass layer.
GlassThickness2	*IfcPositiveLengthMeasure*	Thickness of second (intermediate or outer) glass layer.
GlassThickness3	*IfcPositiveLengthMeasure*	Thickness of the third (outer) glass layer.
FillGas	*IfcLabel*	Name of the gas by which the gap between two glass layers is filled. It is given for information purposes only.
GlassColor	*IfcLabel*	Colour (tint) selection for this glazing. It is given for information purposes only.
IsTempered	*IfcBoolean*	Indication whether the glass is tempered or not.
IsLaminated	*IfcBoolean*	Indication whether glass is layered with other materials.
IsCoated	*IfcBoolean*	Indication whether glass is coated with a material.
IsWired	*IfcBoolean*	Indication whether the glass includes a contained wire mesh to prevent break-in.
Translucency	*IfcPositiveRatioMeasure*	Fraction of the visible light that passes the glazing at normal incidence. It is a value without unit.
Reflectivity	*IfcPositiveRatioMeasure*	Fraction of the visible light that is reflected by the glazing at normal incidence. It is a value without unit.

Figure 10-8. *Pset_DoorWindowGlazingType* property set (thermal and solar transmittance properties not shown).

Glazing properties are available in the IFC Model within *IfcSharedBldgElements* interoperability layer through a *Pset_DoorWindowGlazingType* property set (Figure 10-8, adapted from Adachi et al., 2003) in which only the *GlassThickness* properties

are relevant as a mapping to **BRANZFIRE**. Further material properties may be obtained through the *IfcMaterialProperties* entity sub-types, specifically the thermal conductivity from the *IfcThermalMaterialProperties* entity; and the thermal expansion coefficient and the Young's modulus from the *IfcMechanicalMaterialProperties* entity. The distance to the flame may be determined if the location of the fire can be identified such as by the addition of a *Pset_FurnitureHeatRelease* property set to an item.

10.6 Conclusions

The latest version of the IFC Model supports fire engineering through a number of property sets defined in various places. This chapter has noted some areas where enhancements and extensions to these property sets would be beneficial to fire engineers wishing to use the IFC Model as a source of input data into fire simulation software. It is only through the use of the IFC Model that feedback can be provided to the IFC developers so that they can enrich the fire engineering content of model.

This chapter shows that mappings between the IFC model and the **BRANZFIRE** fire simulation software can be identified, as summarised in Figure 10-9, and these mappings are likely to be appropriate for other fire simulation tools. Progress is already underway, building on the earlier work described elsewhere (Chapter 11), to code the mappings described in this paper into a software tool that would allow the exchange of IFC files with **BRANZFIRE** and to extend the mappings to other fire simulation software tools.

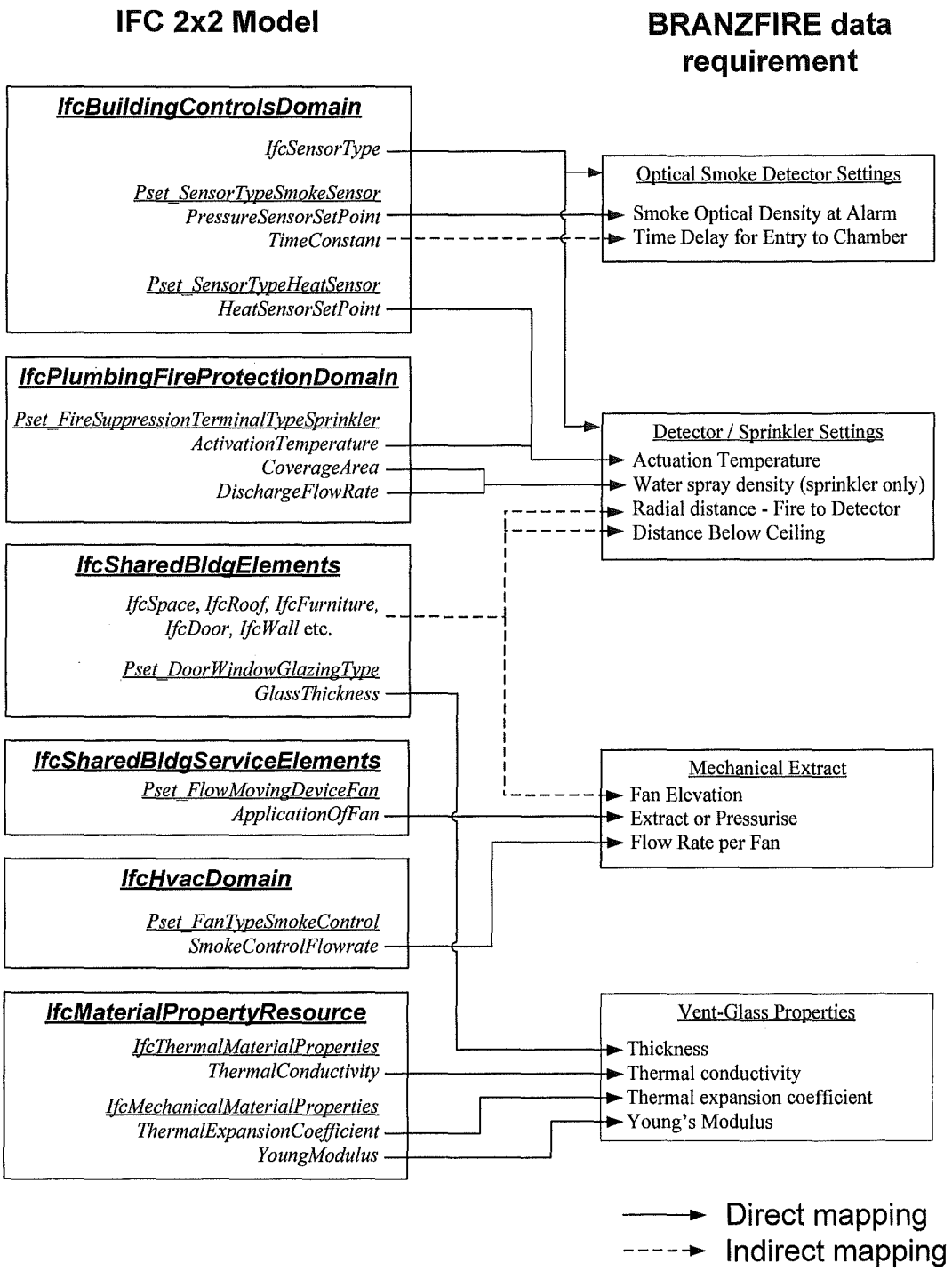


Figure 10-9. Summary of mapping from IFC 2x2 to **BRANZFIRE**.

Chapter 11:
TRANSFER OF ARCHITECTURAL DATA FROM THE IFC MODEL TO A FIRE SIMULATION SOFTWARE TOOL

Spearpoint M J (2005b). Transfer of architectural data from the IFC model to a fire simulation software tool. Submitted to Journal of Fire Protection Engineering, February 2005.

Paper Referees:

Comments yet to be received from two anonymous reviewers

11.1 Abstract

This paper introduces a standardised object-oriented model for buildings that can be used as a means of electronic exchange between various software tools. The ability to transfer architectural data between a commercially available CAD program and a widely available zone fire simulation tool is used to illustrate the applicability of this model in fire engineering.

The paper describes software developed to interpret the model and the test buildings used to verify the exchange process. In general the building geometry, topology and other properties can be transferred satisfactorily but some inconsistencies exist due to the structure of the building model, the CAD implementation of the model and the simplifications required by the zone modelling approach.

11.2 Introduction

The increasing power and availability of personal computers coupled with advances in fire science has meant that the use of fire simulation software tools (commonly referred to as 'fire models') has become more prevalent over the past two to three decades. Prior to execution of a fire simulation software tool, basic information such as the geometry and topology of a building, the location of items within the building and the properties of those items may be required. Depending on the fire simulation software tool being used, the fire performance features associated with the structure, contents or fire protection equipment may also be specified. The input of this information can take a considerable amount of time with the possibility of mistakes and missing information leading to inappropriate output from the simulation tool.

Computer Aided Design (CAD) software is widely used to document the design of a building and the capacity to interpret CAD files into a whole range of simulation software tools is not a new concept. For example, Massa and Cappuccio (1995) concluded that it was readily achievable to adapt a proprietary topological analysis CAD system for modelling bomb blast effects as an interface to fire simulation software although it is unclear whether any subsequent development was undertaken.

In cases where the exchange of CAD data with fire simulation software has already been attempted, efforts have been hindered by the capabilities of the simulation software to interpret CAD data and the ability of CAD systems to adequately describe a building in a way that is meaningful to simulation software. Frost et al. (2001) developed methods to transfer DXF files into the **SMARTFIRE** computational fluid dynamics fire simulation software. Similarly, the **Simulex** evacuation model (Thompson and Marchant, 1995) allows the direct import of CAD data through DXF files. However these methods generally require that the original files be manually 'cleaned' of data that would otherwise be incorrectly interpreted by the intended simulation software. The limitations of formats such as DXF mean that only basic geometric primitives can be derived from the files because more detailed definitions of objects cannot be included (Anon, 2003).

Overcoming the current limitations of data exchange between CAD and fire simulation software requires a much richer description of buildings than traditional CAD systems can provide. Previous work by Mowrer and Williamson (1988) identified object-orientation, the association of attributes with objects and the ability to extract attributes from a CAD-developed drawing database as the key features required by CAD systems to permit integration with fire simulation software. This paper demonstrates how the IFC building product model provides the necessary richness and in particular the ability to exchange its architectural data with a commonly available zone fire simulation tool.

It is not the intention of a building model or the exchange process to take away the 'engineering' portion of an analysis but to try and allow the fire engineer to focus on the design issues rather than spending time setting up a simulation tool. Electronic data exchange software does not mean that fire simulation tools become 'black-boxes' to be used by those unfamiliar with their appropriateness and limitations.

11.3 Building product models

11.3.1 General description

A 'building product model' is a collection of 'entities' that represent building elements, their relationship and their properties. Entities may be physical objects such as walls, doors, windows, etc. or more conceptual entities such as zones, processes or contractual details etc. Properties associated with entities can be divided into several categories (as discussed in Chapter 7) for instance thermo-physical (such as density, thermal conductivity etc.), derived properties (for example the range of operation of a device), regulatory (the fire resistance rating of an element) or descriptive (colour, shape, style etc.).

Building design and operation can be thought of being split into a number of specific domains including architecture, structural engineering, environmental engineering, building services and facilities management. Each domain has its own specialised needs but in many instances there will be parameters that are common across two or more domains related to buildings. Typically these common parameters are the building geometry and topology, the materials and components used in the construction and the location of the structure within the broad environment. The ability to share this information in an electronic form should lead to more effective building design, construction and management. The potential benefits of using building product models for fire engineering have been discussed previously in Chapter 4.

11.3.2 IFC Model

11.3.2.1 Description

The Industry Foundation Classes building product model or 'IFC Model' began development around 1996 and is an extension of a number of earlier related projects such as COMBINE (Augenbroe, 1994) which aimed to integrate energy efficiency software tools for buildings. The latest version of the IFC Model is 2x Edition 2 (Adachi, 2003; Liebich 2003) referred to as 'IFC 2x2' in this paper, with corrections published in Addendum 1.

The IFC Model supports the generation and sharing of project data through the complete building lifecycle amongst a diverse range of domains. The Model is organised into five layers (Liebich 2003) which consist of:

- Resource layer – the lowest level layer which consists of general purpose concepts or objects that are independent of application which rely on other items in the model for their existence.
- Kernel - provides generalised concepts and determines the model structure and decomposition.
- Core extensions - specialisation of concepts defined in the Kernel.
- Interoperability layer - describes concepts common to two or more domains.
- Domain layer – the highest level layer that provides details for specific domains.

The Model does not attempt to include every aspect of a building as this would likely be too complex and take too long to develop, instead entity types are described at a relatively generic level. The limited scope of the IFC Model is addressed by the use of 'property set definitions' which allow for the extension of these generic level entities. As a result, the IFC Model is being implemented and enhanced in many areas of the construction industry. Examples include concrete structures (Rönneblad, 2003); heating, ventilation and air conditioning systems (Bazjanac and Maile, 2004); timber buildings (Osterrieder et al., 2004) and concrete slab bridges (Yabuki and Shitani, 2003).

11.3.2.2 Structure

The contents of the IFC Model are identified with a specific convention which is followed in this paper and is indicated hereafter in italicised text. Entities are prefixed with *Ifc*; property sets prefixed with *Pset* and enumerated lists of entities or properties with the addition of *Enum*.

The IFC Model is essentially created as a tree structure with a top level 'parent' entity giving access to child entities below. However the object-oriented design of the IFC Model includes the ability for inheritance amongst entities and allows complex relationships to exist between entities. The current 2x2 release of the IFC Model

contains over 600 different entities, almost 300 property sets plus another 300 supporting data types. Abstract entities are used to describe collections of related building elements with specific elements defined as sub-types of the generic abstract entity. Entities have associated attributes which are either required or optional as specified by the IFC Model.

The IFC Model is formulated using the EXPRESS data modelling language as defined by the STEP (Standard for the Exchange of Product model data) standard, ISO 10303 : Part 11 (1SO, 1994). The structure of the Model can be viewed using the EXPRESS-G notation which is a graphical notation for identifying classes, attributes and relationships developed within the STEP standard.

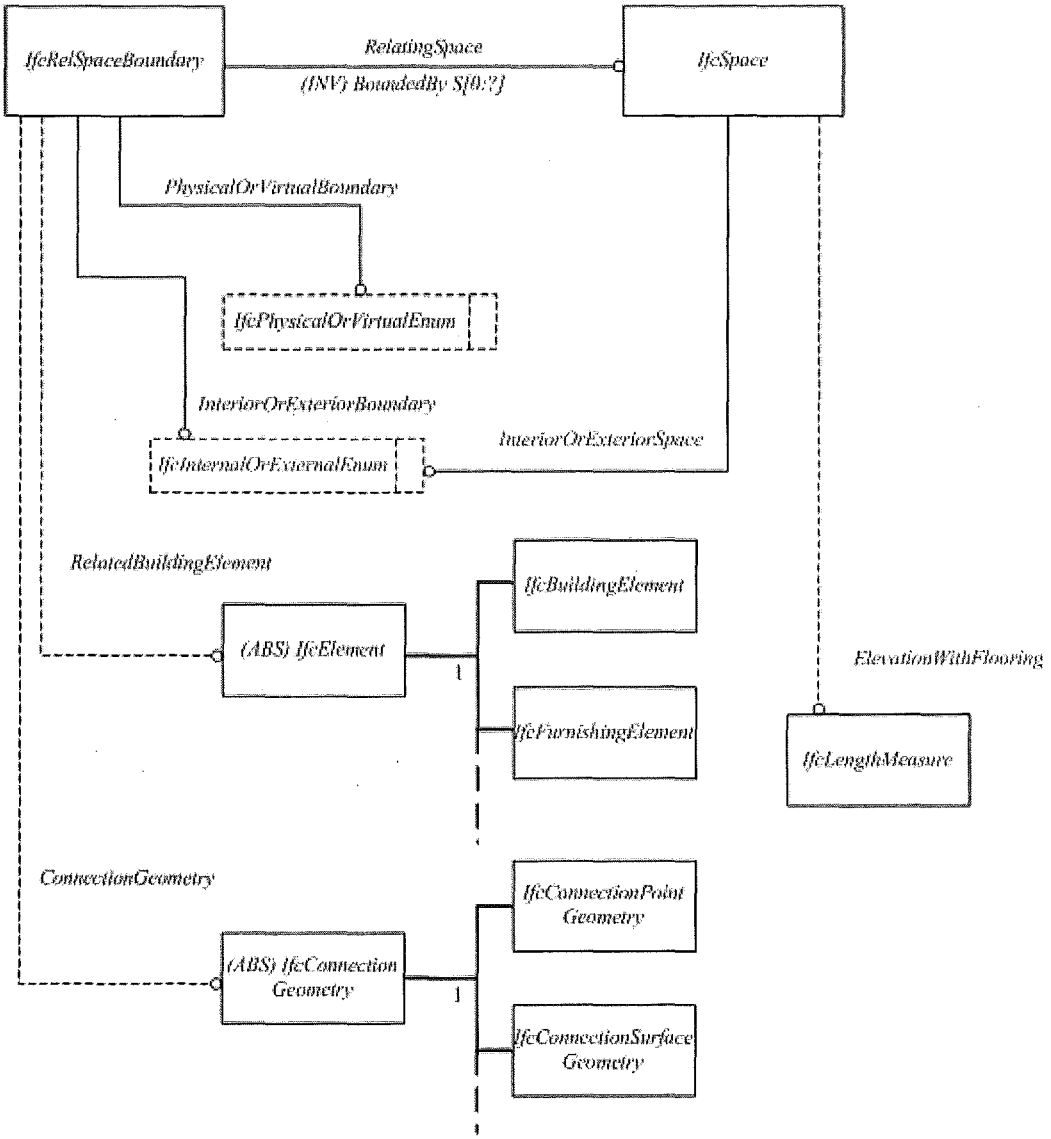


Figure 11-1. EXPRESS-G representation of the ***IfcSpace*** entity and related entities.

Figure 11-1 adapted from Adachi et al. (2003), for example uses an EXPRESS-G diagram to show the *IfcSpace* entity and its relationship with the *IfcRelSpaceBoundary* entity and other connected entities. An *IfcRelSpaceBoundary* entity has an associated *IfcSpace* entity connected by the *RelatingSpace* property and inversely an *IfcSpace* can be *BoundedBy* a set of zero or more *IfcRelSpaceBoundary* entities (denoted by the (INV) and S[0:?] notation respectively). An *IfcSpace* entity can optionally (as indicated by the dotted line) have an *ElevationWithFlooring* property specified as an *IfcLengthMeasure* entity. The *IfcSpace* also must have the *InteriorOrExteriorSpace* property designated which is specified by the

InteriorOrExteriorEnum enumeration. Abstract entities are denoted with the (ABS) prefix and these entities cannot exist in themselves but only as one of their subtypes. For clarity, only examples of the available subtypes of the *IfcElement* and *IfcConnectionGeometry* entities are shown in Figure 11-1. The *IfcSpace* entity also inherits properties from higher level entities such as the *IfcObject*, *IfcProduct* etc. entities although these inheritances are not shown in Figure 11-1.

11.3.2.3 File format and generation

IFC files are primarily exchanged using the STEP encoding specifications given in ISO 10303 : Part 21 (ISO, 2002) although an eXtensible Markup Language (XML) version of the IFC Model has also been made available (Liebich, 2001). The STEP encoding is an electronic data interchange standard which has the aim of completely representing a product over its whole lifetime in a neutral format. The representation includes geometric data and non-geometric data such as properties and costs (King and Norman, 1992). A physical STEP file uses only ASCII characters and is a human readable format although software tools are often used to parse a file.

In this work IFC 2x2 conforming STEP files were generated using the commercially available **ArchiCAD** software (Graphisoft, 2005) from Graphisoft with an IFC 2x2 export module included, also written by Graphisoft. **ArchiCAD** is an integrated architectural design tool that represents buildings using a 'virtual building' model. Building elements such as slabs, walls, doors etc are used to construct a 3-dimensional model of a building. **ArchiCAD** building elements are intelligent objects that have their own associated properties and behaviour. The object-oriented nature of **ArchiCAD** means that the building model is more than the 2-dimensional line representation of a building that is common with traditional CAD systems. The building model allows the user to obtain additional information about the building such as bills of materials. **ArchiCAD** can be used to view a building model not only as plans, elevations and sections but also can generate perspective and virtual reality presentations of the building. Since the building model is stored in a central database, any changes are automatically reflected in all the views.

11.4 Exchange process

11.4.1 IfcSTEP Parser software development

The core exchange software developed in this research, referred to herewith as the **IfcSTEP Parser**, is primarily written in C++ using the Microsoft .NET environment. The current version of the **IfcSTEP Parser** software was built on earlier work in which the previous Release 2 of the IFC Model was accessed through its XML encoding described in Chapter 8. The object-oriented nature of the C++ language enables a set of classes to be developed for selected IFC entities and additional classes can be added as the functionality of the exchange software increases. The reading of STEP files was accomplished by incorporating the **IFC SECOM Server** (SECOM, 2005) into the **IfcSTEP Parser** program. The **IFC SECOM Server** tool includes routines to interrogate and extract elements from a specified STEP file and it can be interfaced to Visual Basic or C++ source code.

The complex nature of the IFC Model means that it is impractical to completely develop the **IfcSTEP Parser** software to process every available entity in the Model. Instead the initial software implementation focuses on the interpretation of the geometry of the spaces, connections between spaces and the basic material properties of geometrical elements such as walls and doors.

In order to interpret a STEP file, the **IfcSTEP Parser** software is written to navigate the IFC Model tree. Figure 11-2 shows how the software identifies the top level *IfcProject* entity and then moves down through the tree extracting the *IfcSite*, *IfcBuilding*, *IfcBuildingStorey*, *IfcSpace* entities and so on. C++ classes defined in the software interpret the EXPRESS relationships associated with each entity, such as those illustrated in Figure 11-1 for the *IfcSpace* entity. Figure 11-2 shows the **IfcSTEP Parser** navigation process through 27 of the IFC Model entities. A dotted arrow is shown where additional entities are accessed deeper in to the tree. The version of the **IfcSTEP Parser** developed for this research is currently able to parse 46 of the IFC Model entities sufficient to extract the relevant architectural information required.

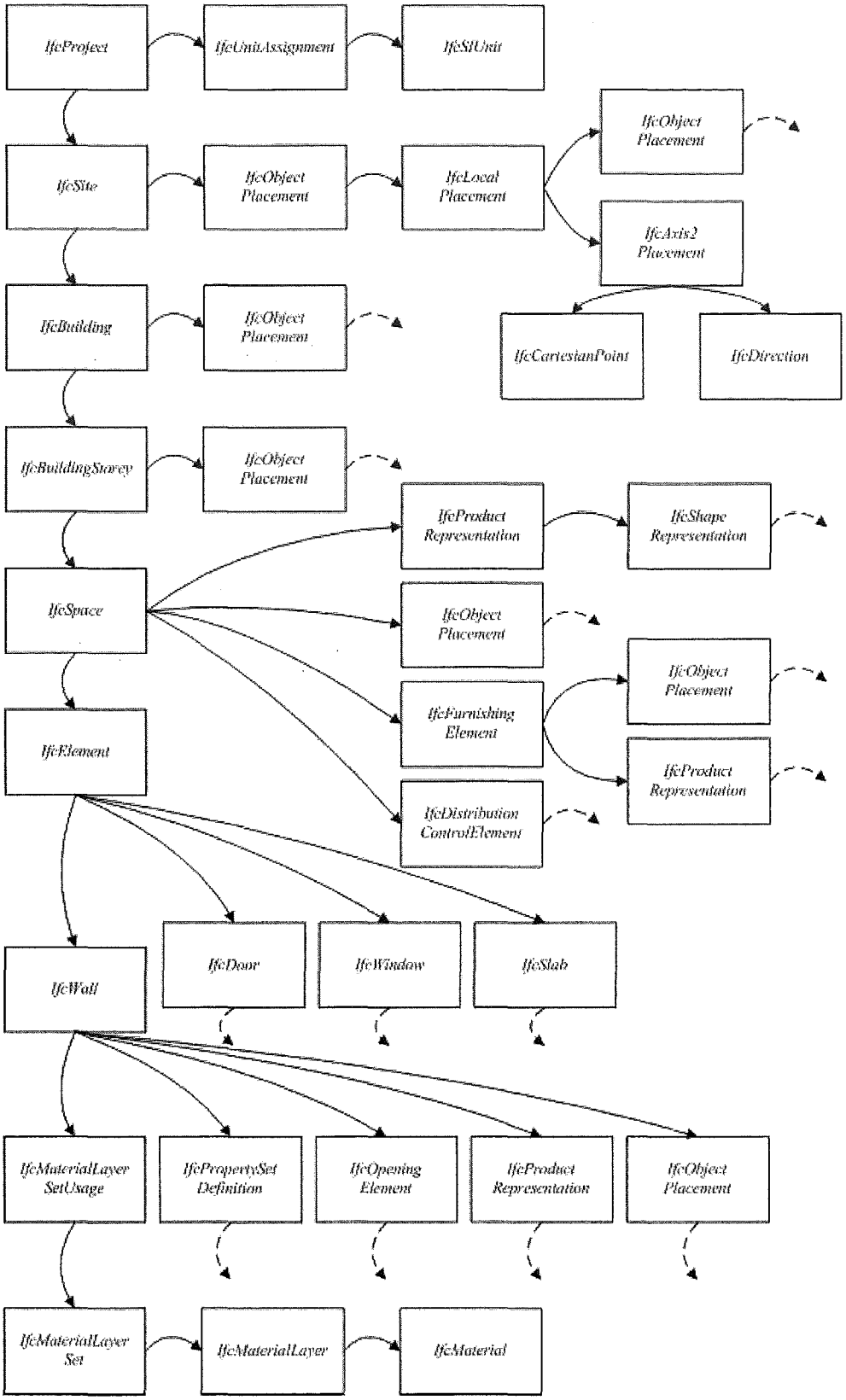


Figure 11-2. IfcSTEP Parser extraction of IFC Model entities (dotted arrows lead to further entities).

The C++ classes interpret the IFC Model into a set of intermediate data structures from which interfaces to specific fire simulation tools can then be developed (Figure 11-3). Each simulation tool interface is developed separate from the core **IfcSTEP Parser** and separate from interfaces to other simulation tools. The advantage of this intermediate approach is that as the IFC Model changes then only the **IfcSTEP Parser** software needs updating. Similarly, if the target fire simulation software tool is modified then only the interface module needs to be adapted.

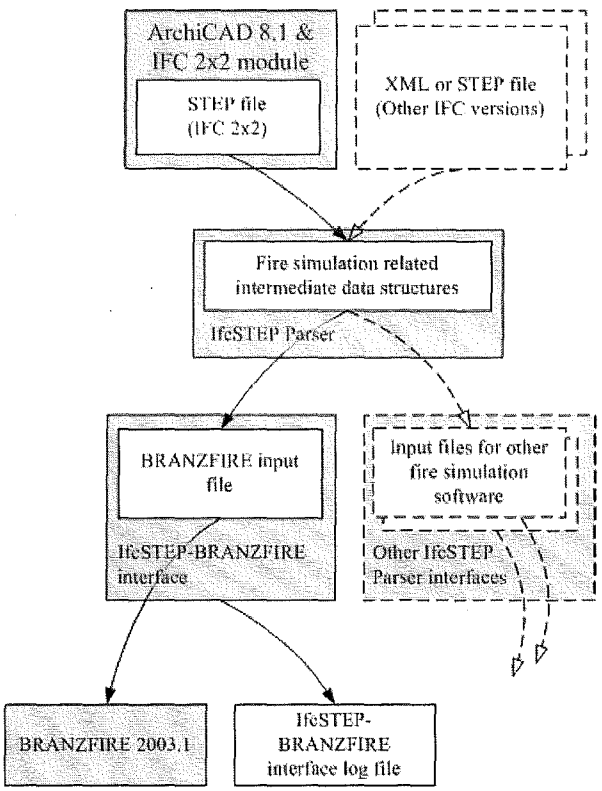


Figure 11-3. IfcSTEP Parser data exchange process (dotted items not currently implemented).

The execution of an **IfcSTEP Parser** interface generates a log file which allows the user to investigate how the interface converted the intermediate internal data structures into the requirements of the target fire simulation software input file. The user can use this log file to track inconsistencies in the exchange process and make subsequent changes to the desired fire simulation software scenario before running any simulations.

11.4.2 BRANZFIRE fire simulation software

In order to demonstrate the ability to exchange IFC Model data using the **IfcSTEP Parser**, **BRANZFIRE** version 2003.1 was selected as the target fire simulation software. The earlier mapping exercise using IFC 2x2 discussed in Chapter 10 showed that IFC 2x2 can lend itself to interpretation by fire simulation models such as **BRANZFIRE** and so an **IfcSTEP-BRANZFIRE Parser** interface was written which transformed the intermediate data structures into a **BRANZFIRE** '.mod' input file.

BRANZFIRE (Wade 2003a, Wade 2003b) is a widely available multi-compartment zone model. It can simulate the movement of smoke between up to 10 inter-connected spaces. Fires are specified by a rate of heat release curve or using a built-in fire spread model in the case of room linings. The model also has the ability to incorporate sprinkler and smoke detector activation, the breaking of window glass (Parry et al., 2003) and the effects of mechanical fans. The underlying conservation equations and relationships that are used by most zone fire simulation software including **BRANZFIRE** are described by Quintiere (2002). The fire plume transports combustion products from a cool lower zone to a hot upper zone and the physical conditions within these zones are considered vertically and horizontally uniform. Fire gases flow through vertical and horizontal openings (often referred to as vents in fire simulation software such as **BRANZFIRE**) in compartment boundaries into neighbouring spaces or the 'outside'.

11.4.3 Project settings

Since **BRANZFIRE** requires geometrical dimensions in metres for building elements and millimetres for items such as material thicknesses it is necessary to substantiate the units being used in the IFC file. The *IfcProject* entity is used to obtain the units being used through the *IfcUnitAssignment* entity (Figure 11-2). The *IfcProject* entity can also be used to extract the project title and add that into the **BRANZFIRE** input file.

11.4.4 Rooms

Zone fire simulation software tools often make a number of simplifications regarding the geometry of the spaces being modelled. For example, it is typical but not

necessarily always the case to assume that spaces have a rectangular foot-print and that ceilings are smooth and horizontal. Some zone fire simulation software tools have a limited number of allowable spaces and/or a limited number of connections such as doors between spaces. In the zone fire simulation software these connections might be assumed to be in the centre of the parent wall rather than at its actual location. Assumptions such as these need to be accounted for when exchanging the IFC Model description of a real building to the idealised view required by zone fire simulation software.

In **BRANZFIRE**, the basic building topology is specified by a collection of inter-connected rooms. The structure of the IFC Model allows each individual room to be identified through the *IfcSpace* entity. The **IfcSTEP-BRANZFIRE Parser** maps *IfcSpace* entities to the **BRANZFIRE** rooms with the *IfcSpace* entity label used to identify the name of the room in the fire simulation input file. The height of a space is obtained from the height of the bounding box of the *IfcSpace* entity. **BRANZFIRE** requires rectangular-shaped rooms and thus spaces with complex footprint geometries will not convert well. Spaces are normally bounded by a wall, open to a neighbouring space or open to the outside and the mapping of these IFC entities are described below.

11.4.5 Walls

As illustrated in Figure 11-1, IFC Model *IfcSpace* entities are associated with one or more *IfcElement* entities using the *IfcRelSpaceBoundary* entity. These bounding entities can be *IfcWall* entities or some other appropriate entity. The **IfcSTEP-BRANZFIRE Parser** identifies the *IfcWall* entities bounding a space and determines its position and dimensions through the *IfcObjectPlacement* and *IfcProductRepresentation* entities respectively (Figure 11-2).

Walls may be constructed of one or more layers each of different materials and thicknesses. Wall layers are obtained through the *IfcMaterialLayerSetUsage* entity that is referenced by an *IfcWall* entity. Each layer thickness is obtained from the *LayerThickness* property associated with the *IfcMaterialLayer* entity and each layer's material type is taken from the *Description* property of the *IfcMaterial* entity as illustrated in Figure 11-2.

Single material layer and multiple material layer walls can be defined in **ArchiCAD**. The **IfcSTEP-BRANZFIRE Parser** maps the *IfcMaterialLayerSetUsage* entity associated with each *IfcWall* entity to **BRANZFIRE** wall materials. The outer most facing *IfcMaterialLayer* entity is assigned to the **BRANZFIRE** wall lining material and thickness. The next *IfcMaterialLayer* entity in the *IfcMaterialLayerSetUsage* entity list is assigned to the **BRANZFIRE** substrate and subsequent *IfcMaterialLayer* entities are ignored since **BRANZFIRE** only handles walls defined with one or two layers.

There are several general issues associated with the translation of bounding entities from the IFC Model to **BRANZFIRE**. An *IfcSpace* generally has more than one bounding wall and these are not necessarily defined as having similar construction. **BRANZFIRE** assumes that a room has a single type of wall surrounding it and this can lead to inconsistencies when mapping from the IFC Model. Where no wall bounding a space has a description then the equivalent **BRANZFIRE** room walls will not be assigned material properties. Where a set of walls bounding an *IfcSpace* have a mix of *IfcMaterialLayer* entities, the **BRANZFIRE** wall materials will be assigned the properties of the 'last' *IfcWall* entity in the set. The 'last' bounding *IfcWall* entity will depend on how **ArchiCAD** outputs its IFC file and thus cannot be necessarily predicted or assumed to be the same each time the file is saved. Furthermore, where multiple layer walls include air gaps it is possible that one of the **BRANZFIRE** wall lining properties (generally the substrate) will be designated as air rather than as a solid material. The user needs to be aware of these issues and check the **BRANZFIRE** scenario setup, such as by viewing the **IfcSTEP-BRANZFIRE Parser** log file, before continuing with any simulations.

11.4.6 Openings

Space boundaries may contain openings such as doorways, windows and holes that connect to neighbouring spaces or the outside. Doorways and windows may have the opening completely filled by the door leaf or the pane of glass. Alternatively there may be an open path through the opening if the door leaf or window is partially or fully open.

The structure of the IFC Model permits the association of an *IfcOpeningElement* with the *IfcElement* abstract entity through the *IfcRelVoidsElement* relationship. This structure allows for the identification of the parent entity for a particular opening (e.g. in which wall do we find a particular door) and also the identification of a group of openings for a specific entity (e.g. which doors form openings in a given wall).

As shown in Figure 11-2, the **IfcSTEP Parser** program identifies all the *IfcDoor* and *IfcWindow* entities associated with each room during the parsing of the *IfcSpace* entities. Similar to the *IfcWall* entity, the position and dimensions of these are determined with the *IfcObjectPlacement* and *IfcProductRepresentation* entities respectively. During the parsing of *IfcWall* entities any *IfcOpeningElement* entities that do not have a parent *IfcDoor* or *IfcWindow* entity are determined. The *IfcOpeningElement* entity associated with *IfcDoor* and *IfcWindow* entities are not processed to avoid duplication of connections. As each *IfcSpace* entity is processed, each vent identified by the **IfcSTEP-BRANZFIRE Parser** is added to global lists of connection types. These lists are used to create the connection map between spaces as described below.

11.4.7 Connection map

Once each *IfcSpace* entity has been processed the **IfcSTEP-BRANZFIRE Parser** builds a connection map that identifies the connecting vents between rooms. The method in which each space is processed results in connecting elements, such as internal doors, appearing twice in the list of connections: once as a connection between space A and space B; and then again as a connection between space B and space A. Thus multiple connection instances are removed from the processing list before the final connection routes are determined. Connections with only one associated space are assumed to connect with the 'outside'. This is the case for most windows and for doors located on external walls. If a user has only selected a portion of a larger building the connections to spaces not in the selected building portion will be treated as outside connections.

In the current version of the **IfcSTEP-BRANZFIRE Parser** all connections are treated as fully open. In future developments of the parsing tool it should be possible to close off connections and then provide properties for the closing element such as

window systems since the IFC Model includes properties for glazing through the *Pset_DoorWindowGlazingType* property set.

11.4.8 Floors and ceilings

BRANZFIRE generally requires that rooms be given a floor and ceiling material type and thickness similar to walls. In a multi-storey building it is possible that a single building element may represent both a ceiling and a floor and these needs to be considered when interpreting a building model.

The IFC 2x2 Model does not explicitly contain ceiling or floor entities and one simple approach is to represent the construction of floors and ceilings using *IfcSlab* entities. The IFC Model does define an *IfcRoof* entity which is an assembly that groups together related entities that make up the roof such as slabs (*IfcSlab*), rafters and purlins (using the *IfcBeam* entity), or other included roofs, such as dormers which would be also specified with an *IfcRoof* entity. Nevertheless, properties of floors and ceilings can be allocated using property sets included in the IFC Model which are associated with the *IfcCovering* entity. The *IfcCovering* entity is used to cover some part of another element such as an *IfcSlab* entity.

ArchiCAD slab elements can be used to represent the construction of floors and ceilings. Similar to walls, slabs can be allocated material composites with layers of different material types and thicknesses. The **IfcSTEP-BRANZFIRE Parser** needs to determine what a slab represents in an IFC file so that it can correctly construct the **BRANZFIRE** input file. The **IfcSTEP-BRANZFIRE Parser** uses the position of the slab compared with the dimensions of the associated *IfcSpace* entity. Slabs located at the highest vertical dimension of a space are assumed to be a roof slab and slabs located at the lowest vertical dimension of a space are assumed to be a floor slab. However this method will not be able to identify elements such as suspended ceilings that may be contained with a space.

An alternative approach is for the user to define slabs on particular **ArchiCAD** drawing layers and currently the **IfcSTEP-BRANZFIRE Parser** will interrogate the *PSet_Draughting* property set associated with an *IfcSlab* entity and assume that slabs on a layer containing the word 'Ceiling' is a ceiling and slabs on a layer containing

the word 'Floor' is a floor. Thus the user needs to ensure that specific drawing layers exist in their CAD environment and that slabs are placed on those specific layers. Although this approach is somewhat more flexible than the previous method it contravenes an overall philosophy of providing building representations in a form that does not require any user-specific configurations.

Complex floor or ceiling geometries will not convert well in the current version of the **IfcSTEP-BRANZFIRE Parser** and better algorithms for the identification of floors and ceilings need to be developed.

11.5 Test buildings

Two simple test case buildings were created in order to verify the effectiveness of the translation process. In each case a virtual building model was setup in **ArchiCAD 8.1** and was saved as an IFC 2x2 file directly from **ArchiCAD** without any changes or 'cleaning' necessary. The IFC file was then processed by the **IfcSTEP-BRANZFIRE Parser** and the resultant `.mod` file was input into **BRANZFIRE**. The `.mod` file generated by the **IfcSTEP-BRANZFIRE Parser** was written for **BRANZFIRE** 2002.7 for backwards compatibility with previous releases of **BRANZFIRE**.

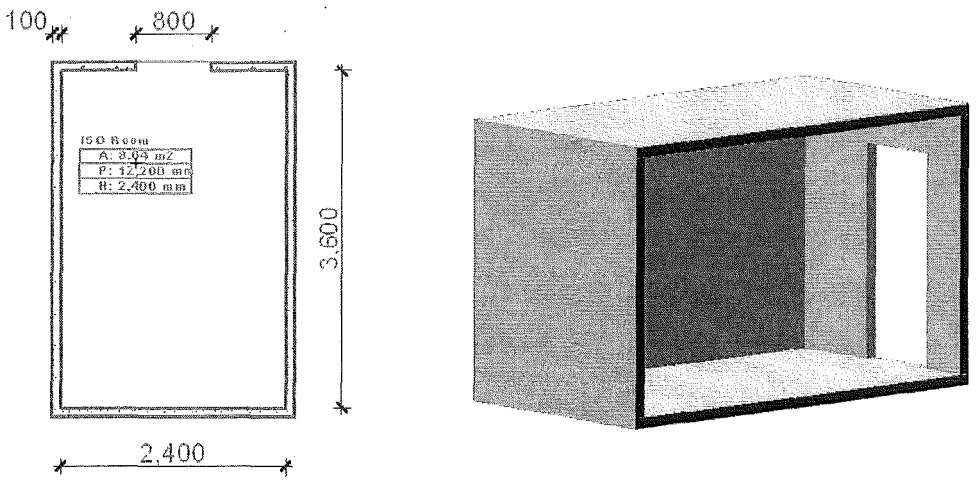
11.5.1 Room/Corner test

A representation of the ISO 9705 (ISO, 1993) room was used as initial test case building. This simple structure was used to check that the basic room geometry and material properties could be extracted from an IFC file. Figure 11-4 shows the **ArchiCAD** drawing of the ISO room with the internal dimensions and door dimensions as specified by ISO 9705. The walls of the test room were set in **ArchiCAD** to 100 mm thick concrete since ISO 9705 only requires that the room be constructed of non-combustible material with a minimum thickness of 20 mm and a density of 500 kg m^{-3} to 800 kg m^{-3}.

ArchiCAD slab elements were used to represent the floor and ceiling of the room. The slab elements were located at highest and lowest vertical room dimensions and also placed on the 'Floors: Slabs' and 'Roof: Ceiling Slabs' **ArchiCAD** default drawing layers respectively. This allowed the identification of the room floor and ceiling by

either of the two methods described earlier. The properties of the ceiling slab was also specified as 100 mm thick concrete whilst the floor slab was set to a composite of 2 mm plaster over 100 mm concrete so as to test the ability of the **IfcSTEP Parser** to identify entities consisting of layers of materials.

Walls can be created in **ArchiCAD** in such a way as to represent the boundaries of a space however these will not explicitly define the contained space. Each space in a building representation must be assigned an *IfcSpace* entity for the **IfcSTEP Parser** to correctly interpret an IFC file. In **ArchiCAD** the 'zone tool' is used to identify separate areas in a building and these areas are exported as *IfcSpace* entities in an IFC file. The **ArchiCAD** zone tool also allows the user to specify the characteristics of the space and also automatically presents information such as the net floor area, perimeter and height on the drawing plans as illustrated in Figure 11-4 (a).



(a) (b)

Figure 11-4. ISO 9705 test room (a) plan elevation, (b) cut-through isometric view.

After importing the resultant .mod file into **BRANZFIRE** Figure 11-5 shows the **BRANZFIRE** *Room Design* windows which are utilised by the user to define room parameters. Figure 11-5 (a) shows that the correct room dimensions have been determined and Figure 11-5 (b) shows the door (wall vent) has also been properly extracted. The wall material and thickness is shown in Figure 11-5 (c) and the composite floor has been correctly interpreted in Figure 11-5 (d). Note that the

material names are those specified by the **ArchiCAD** material list and not from the **BRANZFIRE** database.

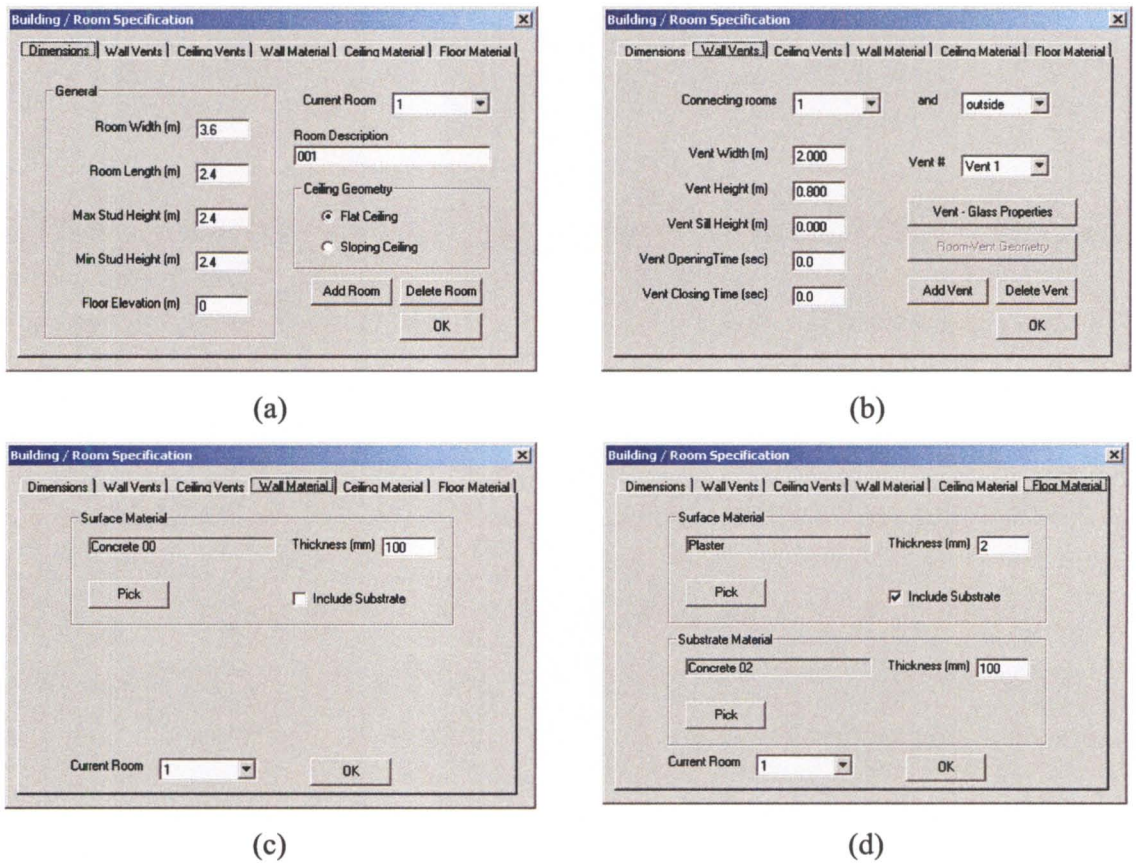


Figure 11-5. BRANZFIRE *Room Design* windows for the ISO 9705 test building (a) room dimensions; (b) wall vent; (c) wall material (d) floor material.

11.5.2 Cardington house

Figure 11-6 shows the ground floor of a residential house based on the geometry of a two-storey building which has been used for various fire engineering-related studies (Spearpoint and Smithies, 1999). The end elevation and isometric view were automatically generated from **ArchiCAD**'s virtual building model. The second storey of the house was omitted because the current version of the **IfcSTEP-BRANZFIRE Parser** does not process *IfcStairs* entities. The stairs were therefore not included in the hallway of the virtual building model.

The purpose of setting up this building was to verify the topological and geometrical translation process within the **IfcSTEP-BRANZFIRE Parser**. The geometry of the spaces was kept simple with rooms represented by rectangular footprints. For the purposes of the verification process, no roof or floor slabs were included in the virtual building model and material properties for building elements were minimal.

The house has several inter-connected rooms with one or more connections. Connections are either internal standard-size doors or an opening in a wall plus doors and windows on the external boundaries including the large garage door.

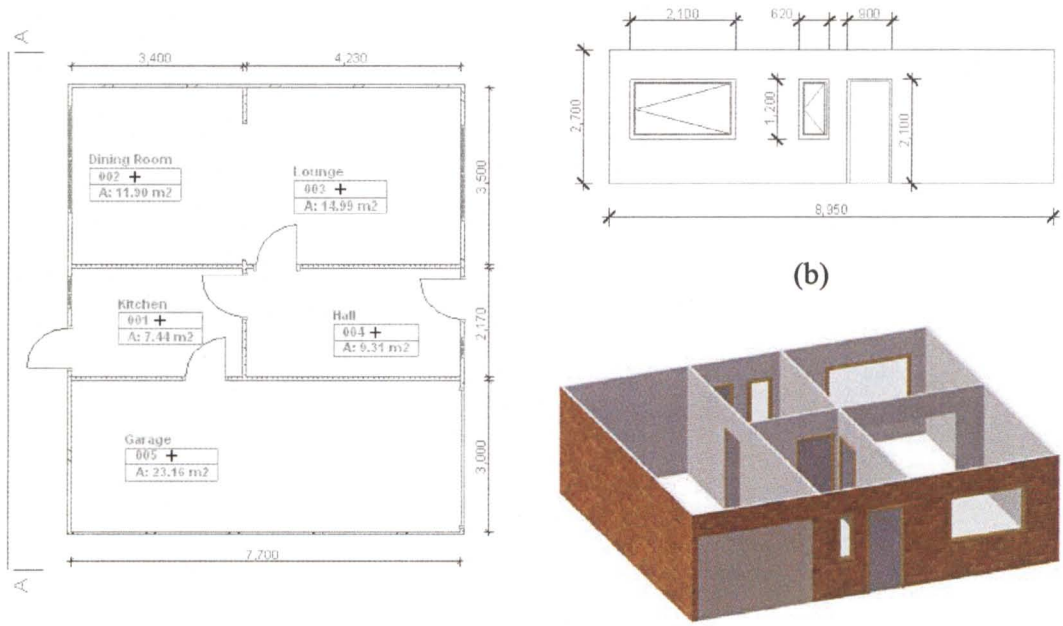


Figure 11-6. Ground floor of the two-storey residential building (a) plan elevation, (b) end elevation, section A-A, (c) isometric view.

Figure 11-7 shows the log file output from the **IfcSTEP-BRANZFIRE Parser** and it can be seen that the dimensions of rooms, doors, windows and openings have all been properly interpreted. Furthermore, the connections between rooms have been correctly identified. It should be noted that in this example it has been assumed that all vents are fully open since the purpose of the exercise is to specifically test the connection mapping functionality of the **IfcSTEP-BRANZFIRE Parser**. The ability to assign

descriptive labels to building elements was not utilised in **ArchiCAD** with the result being door, window and opening names are listed as '$' in the log file.

Material properties for the walls have been determined where 'Double 1:8' is the style of brickwork specified for the outside walls. Note how some spaces have their walls set to 'Plaster' whilst others are 'Double 1:8'. This is a result of rooms having walls assigned different properties and the **IfcSTEP-BRANZFIRE Parser** using the 'last' wall in the IFC file to define the wall properties for the whole space.

```
Log file for IfcSTEP-BRANZFIRE Interface
version 0.1 (Jan 20 2005)
Started at 16:46:49 on Thursday 20 January 2005

**********************************************

IfcSTEP-BRANZFIRE Interface
---------------------------

(c) Michael Spearpoint
University of Canterbury
New Zealand

Interface version 0.1 (Jan 20 2005)
IfcSTEP Parser version 0.1
BRANZFIRE version 2002.7

**********************************************


Space '001' created as BRANZFIRE room 1
      width = 2.17m  depth = 3.40m  height = 2.70m
      Wall lining thickness set to '70.00'
      Wall lining material set to 'Plaster'

Space '002' created as BRANZFIRE room 2
      width = 3.40m  depth = 3.50m  height = 2.70m
      Wall lining thickness set to '70.00'
      Wall lining material set to 'Plaster'

Space '003' created as BRANZFIRE room 3
      width = 3.50m  depth = 4.23m  height = 2.70m
      Wall lining thickness set to '70.00'
      Wall lining material set to 'Double 1:8'

Space '004' created as BRANZFIRE room 4
      width = 2.17m  depth = 4.23m  height = 2.70m
      Wall lining thickness set to '70.00'
      Wall lining material set to 'Plaster'

Space '005' created as BRANZFIRE room 5
      width = 3.00m  depth = 7.70m  height = 2.70m
      Wall lining thickness set to '70.00'
      Wall lining material set to 'Double 1:8'

Door '$' created as BRANZFIRE vent
      width = 0.90m  sill = 0.00m  soffit = 2.10m
      connecting from room 1 (001) to room 5 (005)

Door '$' created as BRANZFIRE vent
      width = 0.90m  sill = 0.00m  soffit = 2.10m
      connecting from room 1 (001) to room 4 (004)

Door '$' created as BRANZFIRE vent
      width = 0.90m  sill = 0.00m  soffit = 2.10m
      connecting from room 1 (001) to outside

Door '$' created as BRANZFIRE vent
      width = 0.90m  sill = 0.00m  soffit = 2.10m
      connecting from room 3 (003) to room 4 (004)

Door '$' created as BRANZFIRE vent
      width = 0.90m  sill = 0.00m  soffit = 2.10m
      connecting from room 4 (004) to outside

Door '$' created as BRANZFIRE vent
      width = 2.80m  sill = 0.00m  soffit = 2.10m
      connecting from room 5 (005) to outside

Window '$' created as BRANZFIRE vent
      width = 0.62m  sill = 0.90m  soffit = 2.10m
      connecting from room 1 (001) to outside

Window '$' created as BRANZFIRE vent
      width = 2.10m  sill = 0.90m  soffit = 2.10m
      connecting from room 2 (002) to outside

Window '$' created as BRANZFIRE vent
      width = 1.90m  sill = 0.90m  soffit = 2.10m
      connecting from room 3 (003) to outside

Window '$' created as BRANZFIRE vent
      width = 0.42m  sill = 0.90m  soffit = 2.10m
      connecting from room 4 (004) to outside

Opening '$' created as BRANZFIRE vent
      width = 2.70m  sill = 0.00m  soffit = 2.10m
      connecting from room 2 (002) to room 3 (003)
```

Figure 11-7. IfcSTEP-BRANZFIRE log file output (in two column format).

11.6 Discussion

11.6.1 Efficiency gains

The objective of using a standardised model of a building to obtain data for simulation software is an increased level of efficiency and whether this has been achieved is a reasonable question. On the positive side this work shows that it is possible to obtain geometrical and topological data from rooms with a simple shape. However, where rooms are complex in shape or in the properties associated with the building elements, the user still is likely to have to intervene before executing their simulations. This raises the issue of whether the effort to create a virtual building model in a tool such as **ArchiCAD** is worth it compared to directly setting up scenarios in the target fire simulation tool.

Fire simulations conducted by a fire engineer invariably require some form of graphical record of the building being analysed. An **ArchiCAD** virtual building model provides such a graphical representation that can be used to communicate with others in a written or oral report. Furthermore, where the building model has already been created by someone else, such as the architect, it still saves the manual re-entry of basic geometrical information even if some further manipulation is necessary. The benefits of using a common electronic building description become greater when multiple simulation tools are considered. Changes to the building model are reflected in all subsequent runs of any simulation tool rather than each change being individually incorporated into each simulation tool.

11.6.2 Future work

The complexity and scope of the IFC Model means that there are still considerable enhancements that can be added to the **IfcSTEP Parser** software. The current version can only handle less than ten percent of the entities available in IFC 2x2. However it should be recognised that there may be many of the IFC entities which will have no direct use in fire simulation software.

In the longer term it is envisaged that IFC Model exchange with other commonly used fire simulation software will be created such as computational fluid dynamics, people movement and structural fire analysis tools. There is also the need to continue testing the capabilities of the parsing algorithms using additional buildings. Complex

buildings such as multi-storey structures, those with non-rectangular footprints and composite material properties would all challenge the current version of **IfcSTEP Parser** and would almost certainly require modifications to the program.

By gaining access to the source code of commonly available fire simulation software tools it might be possible to make the translation of an IFC Model file integral with the fire simulation software rather relying on a separate parser tool. However this may require considerable effort in order to integrate the parsing processes particularly where the original fire simulation source code was written some time ago or in a language other than those found in Microsoft's .NET environment.

Finally, the current focus of the **IfcSTEP Parser** development has been to demonstrate the viability of using the IFC Model as an electronic description of buildings. As a result very little effort has been undertaken to develop a user-friendly interface. The current version executes as a simple command line program. The Microsoft .NET environment includes tools to build Windows compliant interfaces and the core source code for the **IfcSTEP Parser** is written in a way which enables a suitable front-end to be added reasonably easily.

11.7 Conclusion

The paper demonstrates how the IFC building product model can be used to transfer electronic building descriptions between a commercially available CAD system and the **BRANZFIRE** fire simulation tool. The transfer process is not without its limitations which stem from a combination of the current content of the IFC Model, the implementation of the IFC Model in a CAD package, the ability to populate and extract entities from an IFC file and the mapping of those entities to a simplified form suitable for zone fire simulation representation. There is much further work required to enhance the capabilities of the exchange software including expanding the scope of IFC entities that can be processed, improving the mapping of the IFC entities to the requirements of zone fire simulation software and enlarging the range of fire simulation software that can interface to the software.

Additional material

The following material significantly expands on sections given in the earlier portion of this Chapter. Examples of the content of input and output files generated by the **IfcSTEP Parser** are presented. A comprehensive description of the **IfcSTEP Parser** software is also given including chunks of source code used in the software.

11.8 EXPRESS and EXPRESS-G

As noted in Section 11.3.2.2, EXPRESS-G (ISO, 1994) is a graphical notation for identifying classes, attributes and relationships developed within the STEP standard. There is a direct relationship between the EXPRESS data definition language and EXPRESS-G. However EXPRESS-G is only a sub-set of EXPRESS and not everything that can be defined in EXPRESS can be drawn in EXPRESS-G. The use of EXPRESS-G in the development of the IFC Model is described elsewhere (Anon, 2005) and readers of the IFC Model documentation are directed towards this document as it outlines the meaning and use of the EXPRESS-G symbols as reproduced in Figure 11-8, taken from Anon (2005).

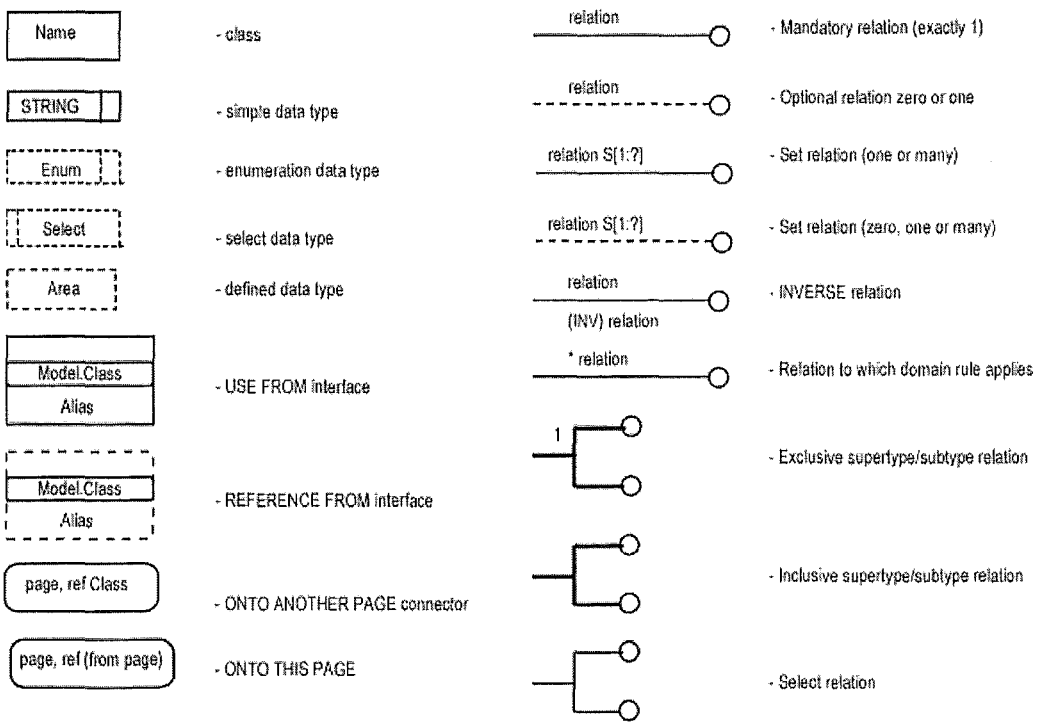


Figure 11-8. EXPRESS-G symbols.

11.9 Exchange process

This section illustrates in more detail the exchange process between **ArchiCAD** and **BRANZFIRE** using the ISO 9705 room (ISO, 1993) representation, presented earlier in Section 11.5.1, as an example. Each step in the process is described below with an example of the files generated where appropriate. Figure 11-9 shows an extended version of Figure 11-3 where again dotted lines indicate items that are not currently implemented. Figure 11-9 indicates where additional log file output takes place during the exchange process. These log files are of particular use to the **IfcSTEP Parser** developer rather than to the end-user of the software.

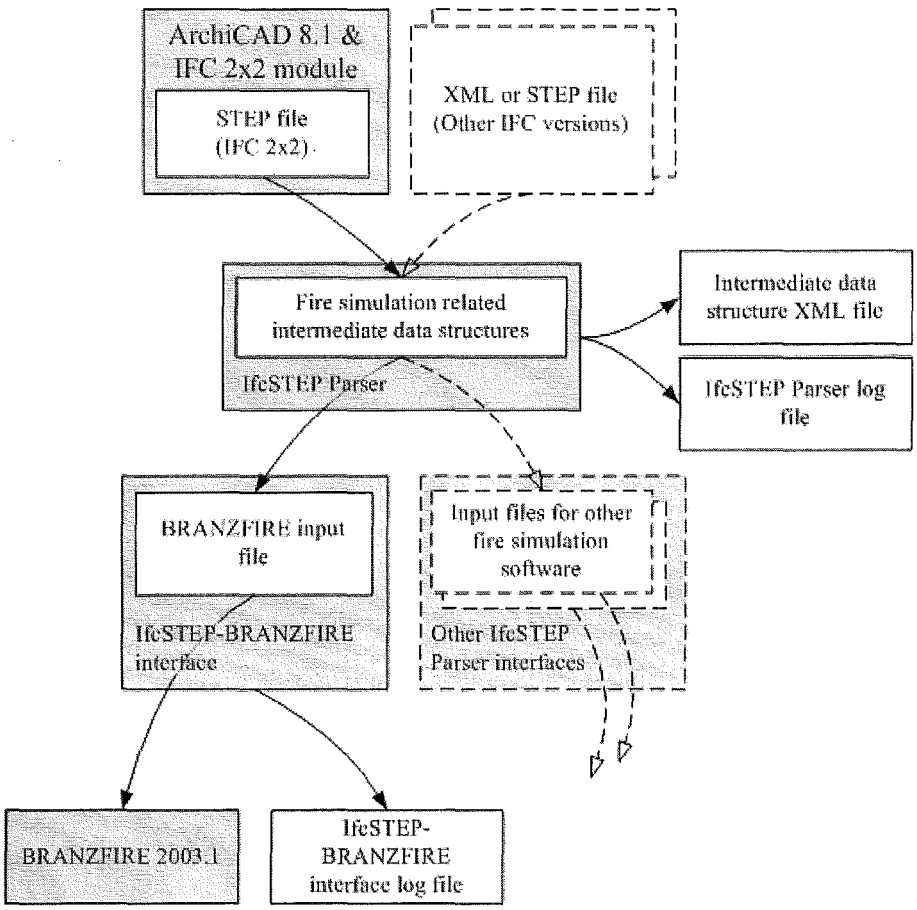


Figure 11-9. IfcSTEP Parser data exchange process – extended version.

11.9.1 IFC file

The export of an IFC file from **ArchiCAD** is a relatively straightforward process. The **ArchiCAD** building model is saved as an .ifc file from the *Save As* menu. Figure 11-10 shows an extract from the IFC file for the ISO 9705 room representation. The complete IFC file for the ISO 9705 room representation contains 538 entity instances and it is not necessary to show the whole set in this thesis. Figure 11-10 shows the standard header required by a STEP file which includes a description of the file properties and the IFC schema used to generate the IFC object instances. The `DATA` portion of the IFC file lists each object with its unique identification number (i.e. '`#xx`' where '`xx`' is an integer from 1 upwards) and then gives the IFC entity name associated with the object. Each object has a list of comma separated properties attached which may be numerical, textual or defined values or cross-references to other objects in the IFC file.

```
ISO-10303-21;
HEADER;
FILE_DESCRIPTION (('ArchiCAD 8.10 Release 2 generated IFC file.',
  'Build Number of the Ifc 2x2      interface: 35251 (16-01-
  2004)\X\0D\X\0A'), '2;1');
FILE_NAME ('ISO9705.ifc', '2005-01-19T17:18:18', ('Architect'),
  ('Building Designer Office'), 'PreProc - Ifc Step Toolbox Version
  2X2 (7. May 2003)', 'Windows System', 'The authorising person.');
FILE_SCHEMA (('IFC2X2_FINAL'));
ENDSEC;
DATA;
#1 = IFCORGANIZATION ('GS', 'Graphisoft', 'Graphisoft', $, $);
#3 = IFCPERSON ($, 'Undefined', $, $, $, $, $, $);
#4 = IFCORGANIZATION ($, 'OrganizationName', $, $, $);
#5 = IFCPERSONANDORGANIZATION (#3, #4, $);
#7 = IFCSIUNIT (*, .LENGTHUNIT., .MILLI., .METRE.);
#8 = IFCSIUNIT (*, .AREAUNIT., $, .SQUARE_METRE.);
#9 = IFCSIUNIT (*, .VOLUMEUNIT., $, .CUBIC_METRE.);
#10 = IFCSIUNIT (*, .PLANEANGLEUNIT., $, .RADIAN.);
#11 = IFCMEASUREWITHUNIT (IFCPLANEANGLEMEASURE (0.0174532925199433),
  #10);
#12 = IFCDIMENSIONALEXPONENTS (0, 0, 0, 0, 0, 0, 0);
#13 = IFCCONVERSIONBASEDUNIT (#12, .PLANEANGLEUNIT., 'DEGREE', #11);
#14 = IFCSIUNIT (*, .SOLIDANGLEUNIT., $, .STERADIAN.);
#15 = IFCSIUNIT (*, .MASSUNIT., $, .GRAM.);
#16 = IFCSIUNIT (*, .TIMEUNIT., $, .SECOND.);
#17 = IFCSIUNIT (*, .THERMODYNAMICTEMPERATUREUNIT., $,
  .DEGREE_CELSIUS.);
#18 = IFCSIUNIT (*, .LUMINOUSINTENSITYUNIT., $, .LUMEN.);
#19 = IFCUNITASSIGNMENT ((#7, #8, #9, #13, #14, #15, #16, #17, #18));
#25 = IFCDIRECTION ((6.123031769111886E-017, 1.));
#6 = IFCOWNERHISTORY (#5, #2, $, .NOCHANGE., $, $, $, 1106108293);
#2 = IFCAPPLICATION (#1, '8.0', 'ArchiCAD 8.0', 'ArchiCAD');
#35 = IFCMATERIAL ('Concrete 00');
#36 = IFCMATERIALLAYER (#35, 100., $);
#37 = IFCMATERIALLAYERSET ((#36), 'Concrete 00');
#38 = IFCMATERIALLAYERSETUSAGE (#37, .AXIS2., .POSITIVE., 0.);
#39 = IFCCARTESIANPOINT ((0., 0.));
#40 = IFCCARTESIANPOINT ((2400., 0.));
```

Figure 11-10. Part of the ISO9705 room representation IFC 2x2 file.

11.9.2 IfcSTEP Parser log file

During the development of the exchange software the **IfcSTEP Parser** can be executed separately from any fire simulation software interface. This allows for checking of the IFC Model interpretation process, the detection of bugs and the identification of areas for enhancement. During parsing the **IfcSTEP Parser** generates a text log file (as shown in Figure 11-9) of the entities it finds so that the programmer can track the progress of the software. Each C++ class has the ability to

set the level of detail that is sent to the log file so that specific areas of progress can be examined in greater depth.

Figure 11-11 shows the log file for the parsing of the ISO 9705 room. As the parser navigates the IFC Model tree each child entity is indented in the log file. The log file also shows a high level of detail for one of the wall entities illustrating the level of depth that the parser can work to for a given entity.

```
Log file for Vanilla IfcSTEP Parser version 0.1 (Jan 25 2005)
Started at 16:41:49 on Tuesday 25 January 2005

*****************************

IfcSTEP vanilla Parser
(c) Michael Spearpoint
University of Canterbury
New Zealand

Interface version 0.1
IfcSTEP Parser version 0.1

*****************************

STEP file loaded
  IfcProject (#27): Default Project
    IfcSite (#29)
      IfcBuilding (#31)
        IfcBuildingStorey (#34) : $
          IfcSpace (#383)  001
              IfcWall (#232)
              IfcWall (#61)
              IfcWall (#108)
              IfcSlab (#334)
              IfcSlab (#288)
              IfcWall (#164)
                IfcObjectPlacement (#163)
                  IfcLocalPlacement (#163)
                  PlacementRelTo #33
                    IfcObjectPlacement (#33)
                      IfcLocalPlacement (#33)
                      PlacementRelTo #30
                        IfcObjectPlacement (#30)
                          IfcLocalPlacement (#30)
                          PlacementRelTo #28
                            IfcObjectPlacement (#28)
                              IfcLocalPlacement (#28)
                                IfcAxis2Placement (#24)
                                  IfcCartesianPoint: 0.00, 0.00, 0.00
                                  IfcDirection (#20)  1, 0, 0
                            IfcAxis2Placement (#24)
                              IfcCartesianPoint: 0.00, 0.00, 0.00
                              IfcDirection (#20)  1, 0, 0
                        IfcAxis2Placement (#32)
                          IfcCartesianPoint: 0.00, 0.00, 0.00
                          IfcDirection (#20)  1, 0, 0
                    IfcAxis2Placement (#162)
                      IfcCartesianPoint: 0.00, 0.00, 0.00
                      IfcDirection (#160)  -1, 0, 0
```

```
IfcProductRepresentation (#159)
  IfcShapeRepresentation (#146)
    IfcPolyline (#145)
      IfcCartesianPoint: 0.00, 0.00
      IfcCartesianPoint: 0.00, 0.00
  IfcShapeRepresentation (#155)
    IfcExtrudedAreaSolid (#154)
      IfcProfileDef (#152)
      IfcArbitraryClosedProfileDef
  IfcShapeRepresentation (#158)
    IfcBoundingBox (#157)
      IfcCartesianPoint: -1.#R, 0.00, 0.00
    Dimensions: 0.00, 0.00, 0.00
IfcPropertySetDefinition (#201)
  IfcPropertySet (#201): Graphisoft AC70 WALL
    IfcComplexProperty (#200) : WALL
      IfcPropertySingleValue (#188) : LAYERNAME
        value: Walls: Exterior   unit: $
      IfcPropertySingleValue (#189) : INFO
        value: Wall-013   unit: $
      IfcPropertySingleValue (#190) : REFMATNAME
        value: Surface-Whitewash   unit: $
      IfcPropertySingleValue (#191) : SIDEMATNAME
        value: Surface-Whitewash   unit: $
      IfcPropertySingleValue (#192) : OPPMATNAME
        value: Surface-Whitewash   unit: $
      IfcPropertySingleValue (#193) : WALL CONTPEN
        value: Pen7   unit: $
      IfcPropertySingleValue (#194) : WALL CONTLTYPE
        value: Solid Line   unit: $
      IfcPropertySingleValue (#195) : WALL CONTPEN3D
        value: Pen2   unit: $
      IfcPropertySingleValue (#196) : WALL FILLPEN
        value: Pen4   unit: $
      IfcPropertySingleValue (#197) : WALL FILLBGPEN
        value: Missing Pen (0)   unit: $
      IfcPropertySingleValue (#198) : WALL USECOMPPENS
        value: 0   unit: $
      IfcPropertySingleValue (#199) : WALL USECOMPBGPEN
        value: 0   unit: $
IfcPropertySetDefinition (#208)
  IfcPropertySet (#208): PSet_Draughting
    PSet_Draughting (#208)
      IfcPropertySingleValue (#203) : Layername
        value: Walls: Exterior   unit: $
      IfcComplexProperty (#207) : Color
        IfcPropertySingleValue (#204) : Red
          value: 74   unit: $
        IfcPropertySingleValue (#205) : Green
          value: 134   unit: $
        IfcPropertySingleValue (#206) : Blue
          value: 0   unit: $
IfcMaterialLayerSetUsage (#142)
  IfcMaterialLayerSet (#141)
    IfcMaterialLayer (#140)
      IfcMaterial (#35) : Concrete 00
    Thickness 0.00
IfcOpeningElement (#178): $


Cleaning up global door list
  0 doors initially in global list
  0 doors remaining in global list


Setting connecting door & spaces: 0 doors in global list
```

```
Cleaning up global window list
  0 windows initially in global list
  0 windows remaining in global list

Setting connecting window & spaces: 0 windows in global list

Setting connecting openings & spaces: 1 openings in global list
```

Figure 11-11. IfcSTEP Parser log file for the ISO 9705 room representation.

11.9.3 Interface log file

When an interface to the **IfcSTEP Parser** is also executed, then a separate log file is generated by the interface module. This log file allows the user to investigate how the interface converted the intermediate internal data structures into the requirements of the target fire simulation software input file. The log file includes a header that records the date that the parser for run and the versions of the interface, **IfcSTEP parser** and intended target simulation tool. The log file then lists details of the spaces and lining parameters where available in the supplied IFC file. Doors, windows and openings between spaces are listed along with their geometry and connection topology.

Figure 11-12 shows the log file for the ISO 9705 representation. Since there is only a single room with a single door, the log file is relatively short. The header lists the date of conversion and the versions of source code used to compile the program. A single space with given dimensions is shown as created as `room 1`. The unnamed (indicted by the '`$`') opening has dimensions as shown and is connected to the outside.

```
Log file for IfcSTEP-BRANZFIRE Interface version 0.1 (Jan 25 2005)
Started at 16:54:24 on Tuesday 25 January 2005

*****************************************

IfcSTEP-BRANZFIRE Interface
---------------------------

(c) Michael Spearpoint
University of Canterbury
New Zealand

Interface version 0.1 (Jan 25 2005)
IfcSTEP Parser version 0.1
BRANZFIRE version 2002.7

*****************************************


Space '001' created as BRANZFIRE room 1
      width = 3.60m  depth = 2.40m  height = 2.40m

Opening '$' created as BRANZFIRE vent
      width = 2.00m  sill = 0.00m  soffit = 0.80m
      connecting from room 1 (001) to outside
```

Figure 11-12. IfcSTEP-BRANZFIRE log file for the ISO 9705 room representation.

On the basis of the contents of the log file the user may want to change the source IFC file and re-parse the file or make manual changes to the software parameters before running a simulation.

11.9.4 BRANZFIRE input file

The result of the exchange process is the creation of an input file appropriate to the target fire simulation software, in this case **BRANZFIRE**. Figure 11-13 shows the final input final for the ISO 9705 representation. It is not the intention to discuss the **BREANZFIRE** input file in detail however it is possible to identify elements within the file that define the room dimensions and basic lining material properties. However, detailed thermo-physical properties are not declared since these properties were not utilised in the **ArchiCAD** building model. The user would need to either select matching materials from the **BRANZFIRE** material database or manually enter appropriate values prior to executing a simulation.

```
"Input file created by IfcSTEP-BRANZFIRE Interface","2002.7"
"Default Project"
"room corner model",0
"Number rooms",1
"Room Number",1
"room width (m)",3.60
"room length (m)",2.40
"room description","001"
"Max room Height (m)",2.40
"Min room Height (m)",2.40
"floor elevation (m)",0.00
"wall lining","none"
"wall substrate","none"
"ceiling lining","none"
"ceiling substrate","none"
"floor substrate","none"
"wall lining thickness (mm)",0.00
"ceiling lining thickness (mm)",0.00
"floor thickness (mm)",0.00
"wall lining conductivity (W/mK)",0.00
"ceiling lining conductivity (W/mK)",0.00
"floor conductivity (W/mK)",0.00
"floor","none"
"wall lining specific heat (kJ/kgK)",0.00
"wall lining density (kg/m3)",0.00
"wall substrate thickness (mm)",0.00
"ceiling substrate thickness (mm)",0.00
"floor substrate thickness (mm)",0.00
"wall substrate conductivity (W/mK)",0.00
"floor substrate conductivity (W/mK)",0.00
"wall substrate specific heat (kJ/kgK)",0.00
"floor substrate specific heat (kJ/kgK)",0.00
"wall substrate density (kg/m3)",0.00
"floor substrate density (kg/m3)",0.00
"floor specific heat (kJ/kgK)",0.00
"floor density (kg/m3)",0.00
"ceiling lining specific heat (kJ/kgK)",0.00
"ceiling lining density (kg/m3)",0.00
"ceiling substrate conductivity (W/mK)",0.00
"ceiling substrate specific heat (kJ/kgK)",0.00
"ceiling substrate density (kg/m3)",0.00
"have ceiling substrate? Yes=-1 No=0",0
"have wall substrate? Yes=-1 No=0",0
"have floor substrate? Yes=-1 No=0",0
"ceiling sloped, 0= flat, -1=sloping",0
"ceiling emissivity",1.0000000
"upper wall emissivity",1.0000000
"lower wall emissivity",1.0000000
"floor emissivity",1.0000000
"interior temp (K)",300.00
"exterior temp (K)",300.00
"relative humidity",0.50
"tenability monitoring height (m)",1.50
"activity level",Light
"radiant loss fraction",0.14
"mass loss per unit area (kg/s)",0.01
"emission coefficient",6.45
"simulation time (s)",0.00
"display interval (s)",0.00
"plume, macaffrey=2, delichatsios=1",2
"suppress ceiling HRR",#FALSE#
```

```
"flame area constant (m2/kW)",0.07
"flame length power",0.67
"burner width (m)",0.17
"wall heat flux (kW/m2)",45.00
"ceiling heat flux (kW/m2)",35.00
"number vents",1
"Room ",1," to ",2," Vent ",1
"vent height (m)",0.80
"vent width (m)",2.00
"vent sill height (m)",0.00
"vent open time (s)",0
"vent close time (s)",0
"glass conductivity(s)",0.00
"glass emissivity(-)",0.00
"glass linear coefficient of expansion (/C)",0.00
"glass thickness (mm)",0.00
"glass shading depth (mm)",0.00
"glass breaking stress (MPa)",0.00
"glass thermal diffusivity (m2/s)",0.00
"glass Young's modulus (MPa)",0.00
"Auto Break Glass",#FALSE#
"Glass fallout time (sec)",0
"Glass to flame distance (m)",0.00
"Glass heated hot layer only?",#FALSE#
"number objects",1
"number data points",0
"energy yield (kJ/g)",0.00
"CO yield (g/g)",0.00
"CO2 yield (g/g)",0.00
"soot yield (g/g)",0.00
"water vapour yield (g/g)",0.00
"Fire height (m)",0.00
"fire location, corner=2, wall=1, centre=0",0
"HRR data"
"Detector Type",0
"RTI",0
"C-factor",0.00
"radial distance (m)",0.00
"actuation temp (K)",0.00
"water discharge rate",0.00
"sprinkler setting",#FALSE#,#TRUE#,#FALSE#
"target radiation endpoint (kW/m2)",1.00
"upper temp endpoint (K)",873.00
"visibility endpoint (m)",10.00
"FED endpoint",1.00
"convective endpoint (K)",353.00
"null.txt"
"null.txt"
"null.txt"
"wall min temp for spread (k)",
"wall flame spread parameter",0.00
"wall effective heat of combustion",0.00
"ceiling effective heat of combustion",0.00
"floor effective heat of combustion",0.00
"fan extract rate (m3/s)",0.00
"fan start time (sec)",0
"fan on?",#FALSE#
"Max Pressure (Pa)",0.00
"Extract?",#TRUE#
"Wall Soot Yield",0.00
"Ceiling Soot Yield",0.00
```

```
"Floor Soot Yield",0.00
"Wall CO2 Yield",0.00
"Ceiling CO2 Yield",0.00
"Floor CO2 Yield",0.00
"Wall H2O Yield",0.00
"Ceiling H2O Yield",0.00
"Floor H2O Yield",0.00
"Floor min temp for spread (k)",0.00
"Floor flame spread parameter",0.00
"fire in room",1
```

Figure 11-13. BRANZFIRE input .mod file generated by the **IfcSTEP-BRANZFIRE** Parser for the ISO 9705 room representation.

11.10 Intermediate data structures

The initial development of the **IfcSTEP Parser** tool employed 'in memory' data as the intermediate structure format. Although this technique was adequate for the task, it was realised that it was difficult for the programmer to visualise these data structures during debugging and enhancements to the **IfcSTEP Parser**. The technique also did not lend itself well to third party development of additional interfaces.

An alternative data structure format was employed during the latter stages of the work in which XML was used to define the intermediate data. This approach allowed the programmer to write the intermediate data to a file (as indicated in Figure 11-9) and then open that file in a browser. The programmer could then view the current status of data structures in a human readable form. The other advantage of this approach is that the schema for the intermediate data structures can be published so that a third party can develop their own interface in parallel with development of the **IfcSTEP Parser** without having to have access to the **IfcSTEP Parser** source code.

Figure 11-14 shows the current 'STEPParser-data-construction' schema generated with the **XML Spy** software (with a complete version of the .xsd file given in Appendix E.1). The XML form of the intermediate data structures has scope for further development beyond the publication of this thesis.

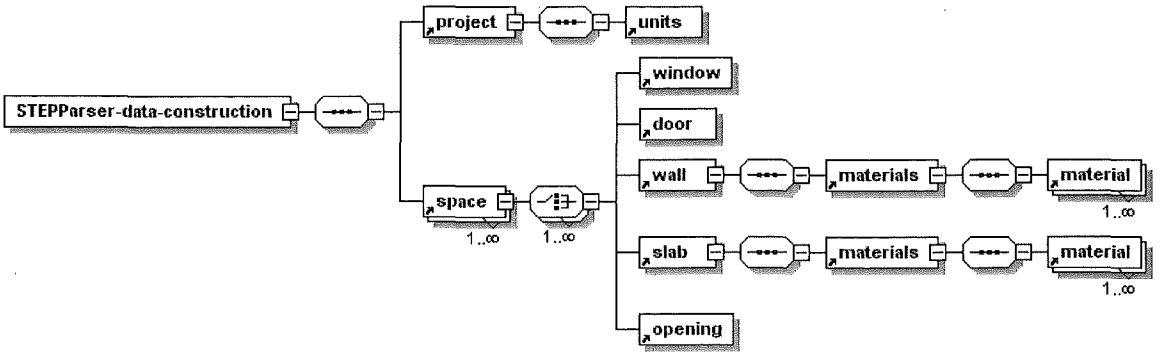


Figure 11-14. Intermediate data structure XML schema.

Figure 11-15 shows the XML file for the ISO 9705 representation conforming to the STEPParser-data-construction schema.

```
<STEPParser-data-construction version="0.1" author="Michael Spearpoint"
      organisation="University of Canterbury" created="11:47:08 on Tuesday 25
      January 2005" xmlns:xsi=http://www.w3.org/2001/XMLSchema-instance
      xsi:noNamespaceSchemaLocation="C:\MJS\Canterbury\Integrated data
      modelling\STEP\Intermediate data\STEPParser-data-construction.xsd">
   <project id="#27" name="Default Project">
      <units id="#19" length="MILLIMETRE" area="SQUARE_METRE"/>
   </project>
   <space id="#383" name="001" width="3600" depth="2400" height="2400"
      pos_x="-2003" pos_y="18696" pos_z="0">
      <wall id="#232" name="" pos_x="-2003" pos_y="14996" pos_z="0">
         <materials id="#211">
            <material id="#210" description="Concrete 00" thickness="100"/>
         </materials>
      </wall>
      <wall id="#61" name="" pos_x="-2003" pos_y="14996" pos_z="0">
         <materials id="#37">
            <material id="#36" description="Concrete 00" thickness="100"/>
         </materials>
      </wall>
      <wall id="#108" name="" pos_x="397" pos_y="14996" pos_z="0">
         <materials id="#86">
            <material id="#85" description="Concrete 00" thickness="100"/>
         </materials>
      </wall>
      <slab id="#334" name="" type="FLOOR" pos_x="-2103"
         pos_y="18796" pos_z="-102">
         <materials id="#340">
            <material id="#337" description="Plaster" thickness="2"/>
            <material id="#339" description="Concrete 02" thickness="100"/>
         </materials>
      </slab>
```

```
        <slab id="#288" name="" type="ROOF" pos_x="-2103" pos_y="14996"
            pos_z="2400">
            <materials id="#291">
                <material id="#290" description="Concrete 00" thickness="100"/>
            </materials>
        </slab>
        <opening id="#178" name="$" width="2000" soffit="800" sill="0" pos_x="1597"
            pos_y="18596" pos_z="0"/>
        <wall id="#164" name="" pos_x="397" pos_y="18696" pos_z="0">
            <materials id="#141">
                <material id="#140" description="Concrete 00" thickness="100"/>
            </materials>
        </wall>
    </space>
</STEPParser-data-construction>
```

Figure 11-15. XML file for the ISO 9705 representation conforming to the STEPParser-data-construction schema.

11.11 Entity restrictions in simulation software

The current version of the **IfcSTEP Parser** has no limit on the number of spaces, connections, boundary materials etc. it can process from an input file but it has already been identified in Section 11.4.4, for example, that **BRANZFIRE** has limits on number of rooms etc. it can handle. Section 8.8.1 and Section 11.4.4 identified limitations with the representation of complex-shaped spaces. Furthermore, Section 8.8.2 and Section 11.4.5 identified where the definition material properties may expose limitations of the mapping between the IFC Model and zone fire simulation software representations. These constraints raise the issues with regard to the quality assurance of the output from fire simulation software tool when using BIM data that has be automatically transferred and how much the fire engineering should be involved in controlling the mapping process.

When considering limits on numbers of entities, one question is where should such limits be dealt with: should it be in the BIM, during the parsing process or should the target simulation software tool check for consistency in the input data? Deleting elements from the BIM conflicts with the desired philosophy of using BIM where the building description, analysis and drawings remain consistent as discussed in Section 1.1. Checking by the target simulation tool would seem a better approach since it would allow the tool to be improved with a corresponding update to the parser

software. However if elements that the fire engineer desires to use in their simulation are not given in the correct location in the input file they might find that the simulation tool has skipped over the necessary elements due to insufficient capacity to deal with them. If the fire engineer does not realise this has occurred then inappropriate output may result from the simulations. A method of tagging elements may provide a suitable solution but requires that there be some way in which tags can be assigned either in the tool that creates the BIM or through some form of interface to the parser.

The limitation on the mapping of complex-shaped rooms may or may not have a major impact on the simulation results. This issue is not limited to this thesis but is a much wider issue for fire engineers. The solution to this will depend on what exactly the fire engineer is trying to achieve. Where the fire engineer is particularly concerned with smoke filling of a space it would be appropriate to approximate a non-rectangular space as a rectangular footprint of equal area. Similarly, where a space has a ceiling that is not horizontal it maybe sufficient to represent the space as having a flat ceiling with a height at the mid-slope height. Alternatively if the issue was the time to operation of a fire detector it is likely that representing a space with a sloping ceiling with a flat ceiling with a height at the mid-slope height might not give appropriate results. In this case it may be more appropriate to use the maximum height of the sloping ceiling as the representation.

Material properties become a possible issue is where a multi-layer wall specification may be needed to obtain flame spread over a wall surface and subsequent heat loss through the wall is to be determined. To calculate ignition and flame spread characteristics a number of properties are required as suggested in Section 7.4 and similar properties are needed to determine heat losses. If only the surface material is defined then it might be found that heat losses are not correctly calculated. If only the substrate material is defined then flame spread may not be appropriately determined. The implications would be more significant where surface and substrate materials have very different fire-related properties.

What this discussion shows is that making decisions about how to map an entity described in a BIM to a fire simulation tool may affect the subsequent value of the

simulation output. The specification of a BIM schema, the interpretation of the schema by a parser developer and its use by the fire engineer all need to be carefully considered.

11.12 IfcSTEP Parser program

A 'vanilla' version of the **IfcSTEP Parser** was written in which no specific fire simulation software was targeted. The vanilla version allowed the author to test the capabilities of the IFC file interpretation and intermediate data structure generation with having to be concerned with the output capabilities.

The core **IfcSTEP Parser** program consists of approximately 12,000 lines of commented C++ source code with around an additional 200 lines for the 'vanilla' module. The volume of code that is used to create the **IfcSTEP Parser** program means it is not practical to include a listing in this thesis or to discuss every component. Instead higher level concepts used to develop the program are discussed and samples of code are given to illustrate some of the details.

11.12.1 Entity processing

The current version of the **IfcSTEP Parser** program has classes available for the following list of 37 IFC 2x2 entities:

IfcAxis2Placement
IfcAxis2Placement3D
IfcBoundingBox
IfcBuilding
IfcBuildingStorey
IfcCartesianPoint
IfcComplexProperty
IfcDirection
IfcDistributionControlElement
IfcDoor
IfcElement
IfcExtrudedAreaSolid
IfcObjectPlacement
IfcOpeningElement
IfcPolyline
IfcProductRepresentation
IfcProfileDef
IfcProject
IfcPropertySet
IfcPropertySetDefinition
IfcPropertySingleValue
IfcShapeRepresentation
IfcSite
IfcSiUnit

IfcFurnishingElement
IfcLocalPlacement
IfcMaterial
IfcSlab
IfcSpace
IfcSpatialStructureElement

IfcMaterialLayer
IfcMaterialLayerSet
IfcMaterialLayerSetUsage
IfcUnitAssignment
IfcWall
IfcWallStandardCase
IfcWindow

Inclusive in these entity classes is code that also processes the following 10 relational IFC 2x2 entities:

IfcRelAggregates
IfcRelAssigns
IfcRelAssociates
IfcRelConnectsElements
IfcRelContainedInSpatialStructure
IfcRelDecomposes
IfcRelDefines
IfcRelFillsElement
IfcRelSpaceBoundary
IfcRelVoidsElement

Other entities that are recognised by the **IfcSTEP Parser** program but not assigned their own class are:

IfcArbitraryClosedProfileDef
IfcRectangularProfileDef

and these entities may have classes created for them in future versions of the **IfcSTEP Parser** program if required.

Figure 11-2 shows a reduced view of the entities processed by the **IfcSTEP Parser**. In this section Figure 11-2 is expanded to show a more detailed view of the processed entities over a series of inter-linked diagrams. Similar to Figure 11-2, Figure 11-16 shows the processing of the top level *IfcProject* entity and subsequent children.

The processing of sub-entities is illustrated in the diagrams that follow. The parsing of the *IfcProductRepresentation* is continued in Figure 11-17, the *IfcObjectPlacement* in Figure 11-18 and the *IfcPropertySetDefinition* in Figure 11-19. The *IfcWall*, *IfcDoor*,

IfcWindow and *IfcSlab* entities are shown in Figure 11-20, Figure 11-21, Figure 11-22 and Figure 11-23 respectively. Finally, the *IfcMaterialLayerSetUsage* entity is shown in Figure 11-24. Dotted lines and rectangles are used to indicate that the processing of child entities continues on a separate diagram.

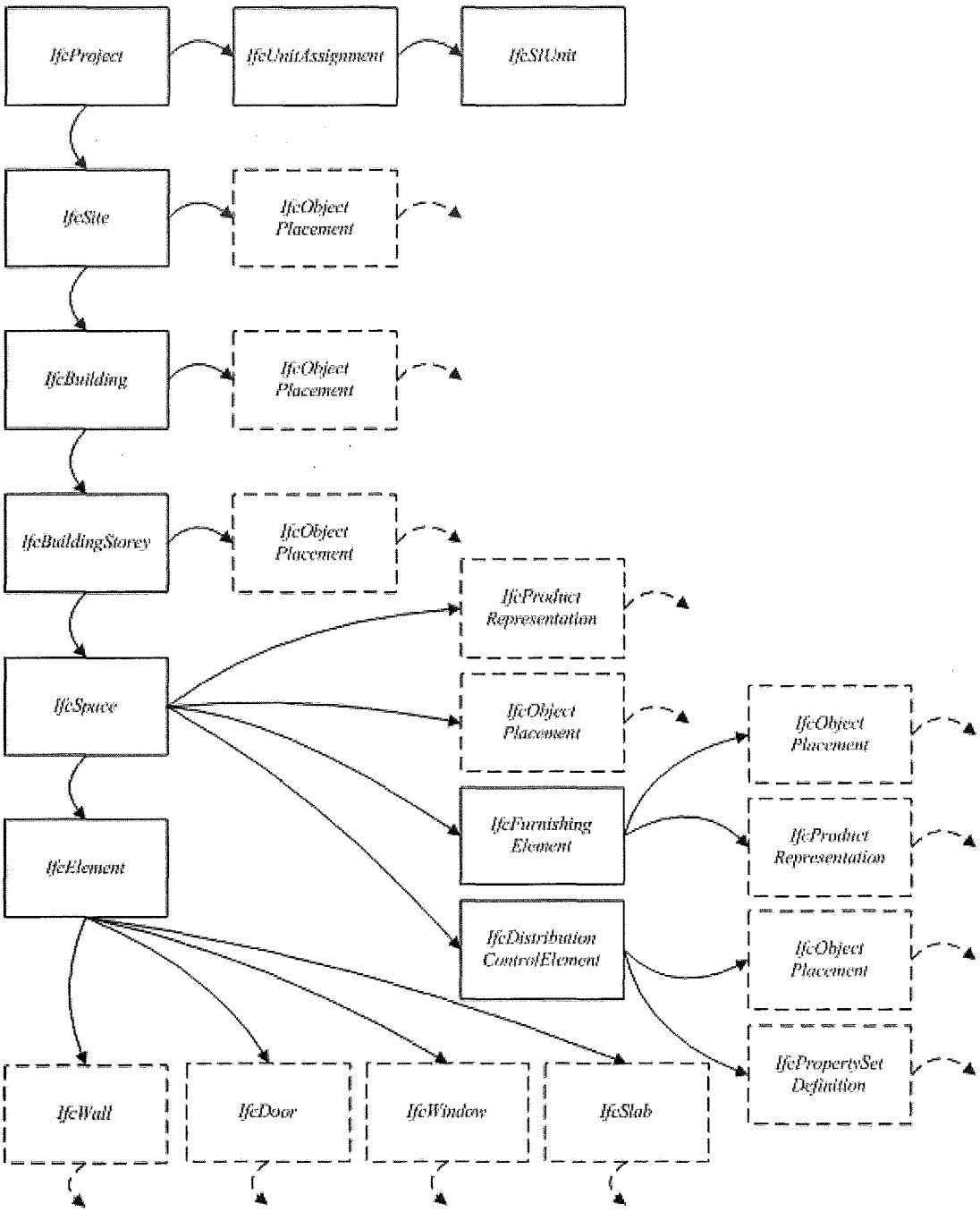


Figure 11-16. IfcSTEP Parser extraction of the top level IFC Model entities (dotted arrows lead to further entities).

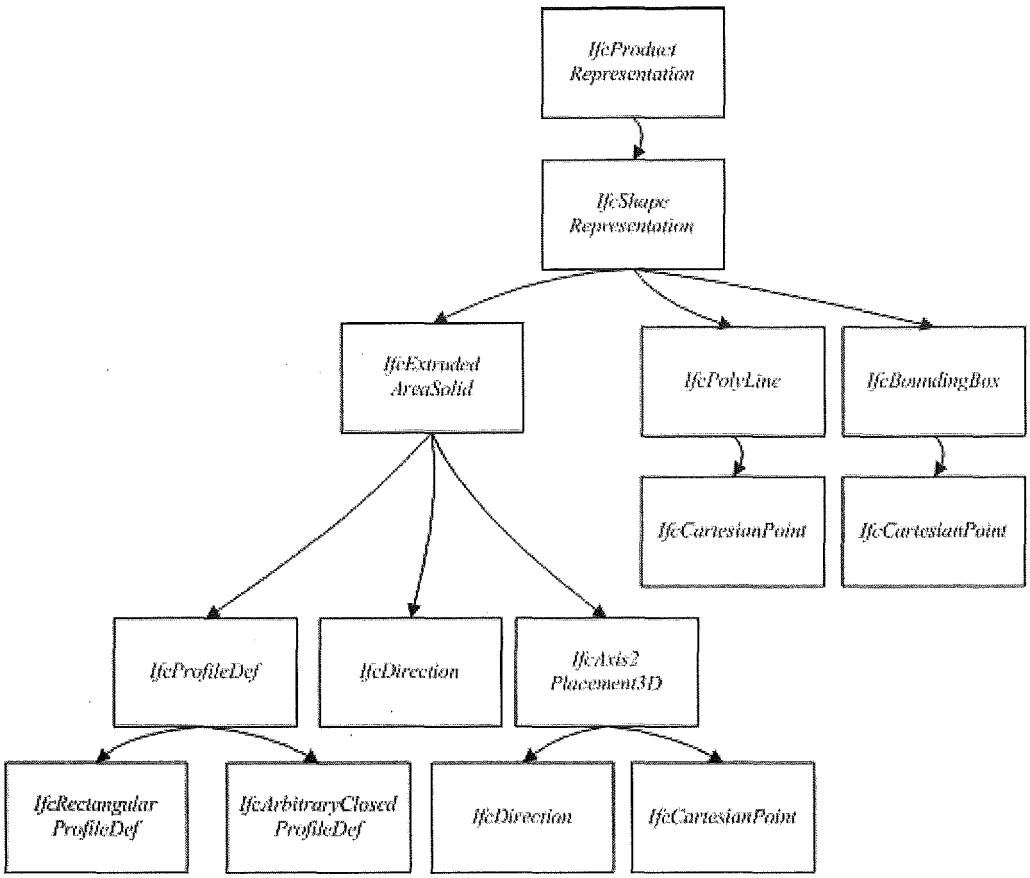


Figure 11-17. The *IfcProductRepresentation* parsing process.

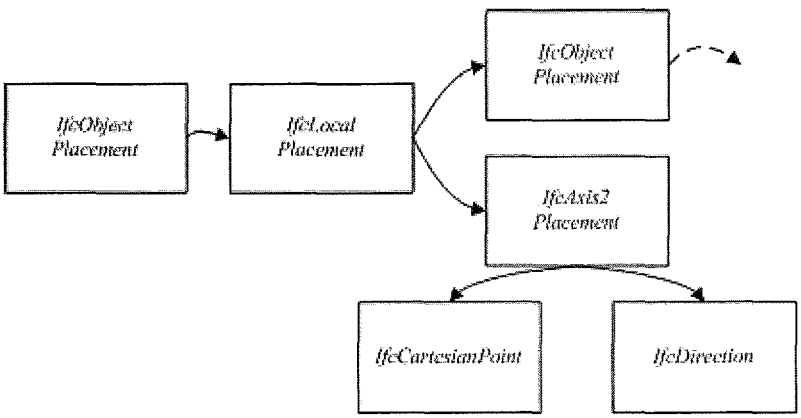


Figure 11-18. The *IfcObjectPlacement* parsing process.

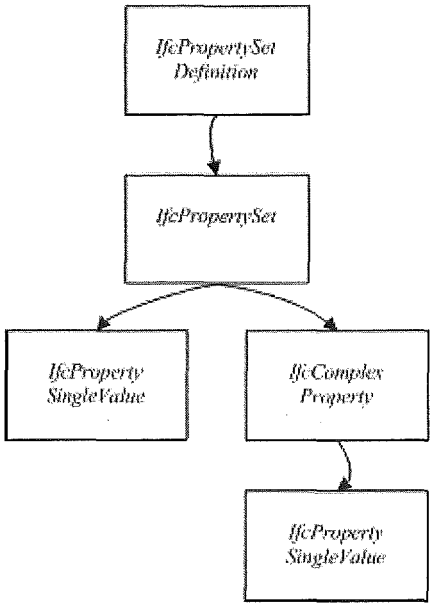


Figure 11-19. The *IfcPropertySetDefinition* parsing process.

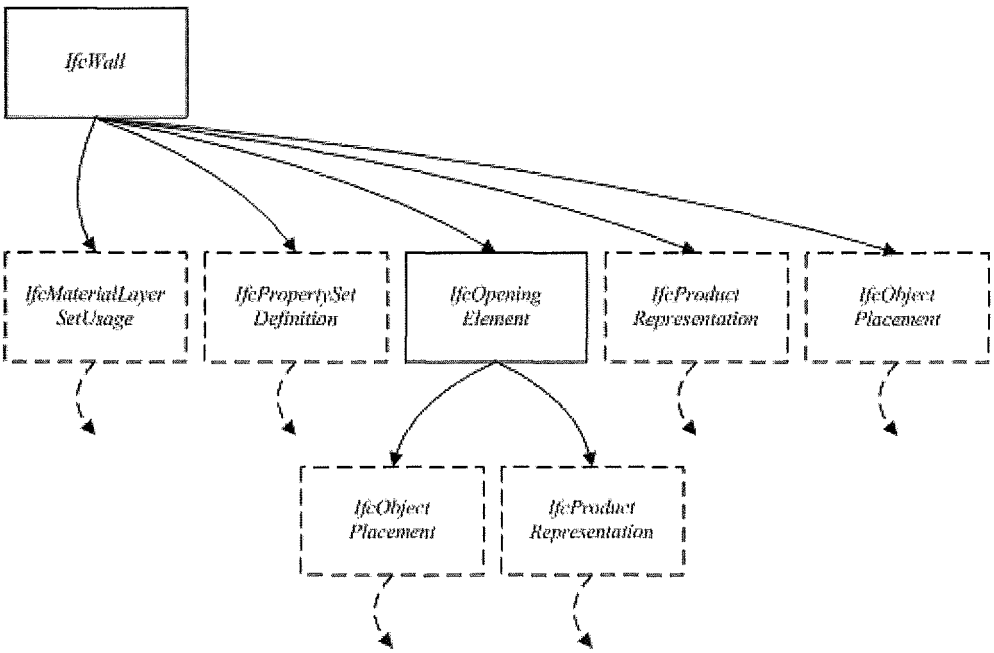


Figure 11-20. The *IfcWall* parsing process.

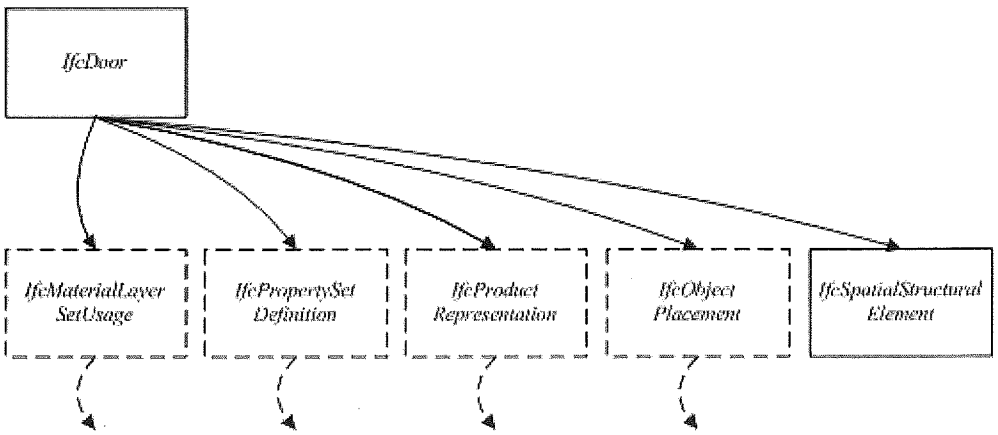


Figure 11-21. The *IfcDoor* parsing process.

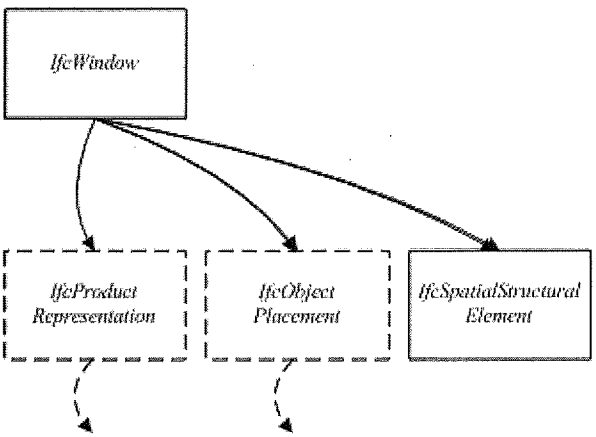


Figure 11-22. The *IfcWindow* parsing process.

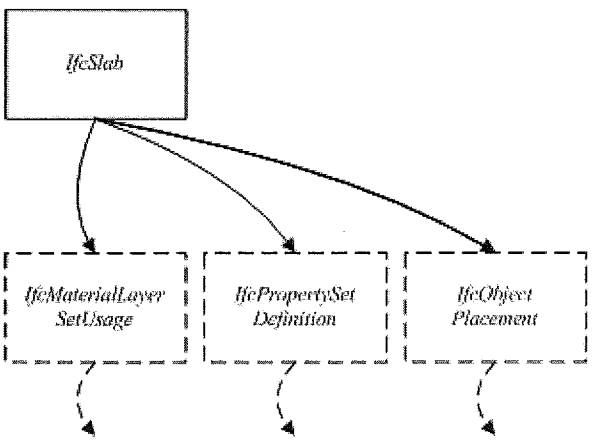


Figure 11-23. The *IfcSlab* parsing process.

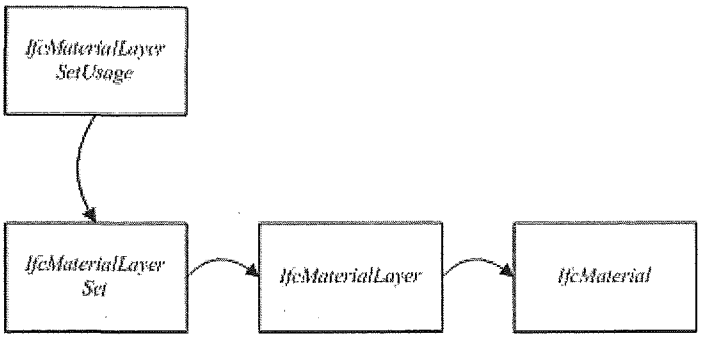


Figure 11-24. The *IfcMaterialLayerSetUsage* parsing process.

11.12.2 Expansion of entity processing

As described in Section 11.12.1 above, the current version of the **IfcSTEP Parser** only handles 37 of the many hundreds of entities defined in IFC 2x2. There is much scope for processing an extended range of IFC entities and this section suggests some candidate entities that might want to be considered in further releases of the **IfcSTEP Parser**.

As described in Section 11.4.5, the **IfcSTEP Parser** currently obtains boundary surface materials from the *IfcMaterialLayerSetUsage* entity associated with each *IfcWall* entity. For interior finishes the *IfcCovering* entity can be used within the IFC Model as an element "which covers some part of another element and is fully dependent on that other element". The IFC documentation (IAI, 2004c) further gives examples of *IfcCovering* elements as "wall, floor and ceiling coverings, finish trim, and base molding". Details for the use of the *IfcCovering* entity are given in the IFC documentation and depend on whether a space has defined boundaries or not. The use of the *IfcCovering* entity would require consideration as to how surfaces are integrated into fire simulation software for future versions of the **IfcSTEP Parser**. Clearly if an *IfcCovering* entity is associated with a boundary then this would need to be used to determine the surface material properties.

In order to integrate smoke control systems and electrical cable performance analysis into fire simulation software some consideration should be made to include building

service element processing into the **IfcSTEP Parser**. Building service elements have their own specific entities defined in the IFC Model. For ducts, entities such as *IfcDuctSegmentType* and *IfcDuctFittingType* are defined in the *IfcHvacDomain* layer so that ducts are broken down into component items. Similarly, cables are broken into segments using the *IfcCableSegmentType* entity in the *IfcElectricalDomain*. The *IfcDuctSegmentType* and *IfcCableSegmentType* entities inherit commonly shared property set definitions and optional product representations from the *IfcFlowSegmentType* entity. It is noted that the *IfcHvacDomain* contains 31 entities and nearly 80 associated property set definitions and the *IfcElectricalDomain* contains 19 entities and around 30 associated property set definitions. It would appear that duct and cabling systems would need to be reconstructed from the component elements and this would potentially require a reasonable effort to integrate then with fire simulation software given the number of entities available to describe these systems.

11.12.3 Source code files

Each C++ class consists of a code file and an associated header file. The code provided for each class corresponds to the EXPRESS structure and consists of one or more methods that process required entity types. Header files provide the scope of individual methods that make up the class. The headers also list other classes and headers that are relied on to compile the class. An example header file is given below (in this case for the *IfcDoor* entity). The original source file is presented in chunks, with each chunk explained in detail.

Chunk 1:

The header begins with a comment block followed by compiler commands that ensure the header is only processed once during compilation.

```
/**********************************************************************/
//
//      S T E P P a r s e r _ I f c D o o r . h p p
//      ===========================================
//
//      Interface for the STEPParser_IfcDoor class.
//
/**********************************************************************/

#if !defined(AFX_STEPPARSER_IFCDOOR_HPP__191D52C6_B3F3_43A4_9263_503C667164A5
      __INCLUDED_)
#define AFX_STEPPARSER_IFCDOOR_HPP__191D52C6_B3F3_43A4_9263_503C667164A5__INCLUDED_

#if _MSC_VER > 1000
#pragma once
#endif // _MSC_VER > 1000
```

Chunk 2:

The header then lists other common header files that are required for the class. These headers contain general type definitions and data structures; access to the standard C++ headers; definitions of global constants and the main `STEPParser` class.

```
// Headers
#include "STEPParser_Common.hpp"
#include "STEPParser_Ifc.hpp"
#include "STEPParser.hpp"
```

Chunk 3:

Forward declarations specify other classes used by the current that are defined in other source files. In this case the `STEPParser_IfcDoor` class using three other classes (for example, see Chunk 5 below).

The `__gc` prefix is used by the complier to specify that the class is automatically garbage collected during runtime. In the Microsoft .NET environment this automatic garbage collection defined as 'managed' code (Simon and Schmidt, 2002). Since the **IfcSTEP Parser** program was initially developed prior to the .NET environment, some of the code runs as 'unmanaged' code and has to be more directly handled by the programmer. It is hoped that eventually the whole of the **IfcSTEP Parser** program will be converted to 'managed' code but this is not essential for the overall functioning of the program.

```
// Forward declarations
__gc class STEPParser_IfcSpace;
__gc class STEPParser_IfcWall;
__gc class STEPParser_IfcBoundingBox;

//////////////////////////////////////////////////////////////////////////
```

Chunk 4:

This section of the header defines the public methods for the class and any inheritance from super-classes. In this case the `STEPParser_IfcDoor` inherits from the `STEPParser_Ifc` class as do all `STEPParser` IFC classes that constitute the **IfcSTEP Parser** program.

Details of the parse method are given in Section 11.12.4. The GetWidth and GetHeight methods return values as STEPParser_measure types which are declared as Int32 in the STEPParser_Common header file. Using definitions for values means it is simple to change a value type throughout the program with a single adjustment to the appropriate type definition. Two static methods are declared in the STEPParser_IfcDoor class which are used to process the connection between spaces via doors. Similar methods are defined for other 'connecting' elements such as the STEPParser_IfcOpeningElement class and the STEPParser_IfcWindow class. Static methods are considered to have class scope and have external linkage which essentially means the methods are not associated specific instances of a class but are globally accessible through the class.

```
__gc class STEPParser_IfcDoor : public STEPParser_Ifc
{
 public:
        // Construction
        STEPParser_IfcDoor();
                                                        // Constructor

        // General functions
        STEPParser_CError *Parse(STEPParser __gc *parseObj,
                IEntity *srcEntity);                    // Parse entity
        STEPParser_measure GetWidth(void);              // Get door width
        STEPParser_measure GetHeight(void);             // Get door height
        STEPParser_CError  *GetConnectingSpaces(void); // Determine connecting spaces

        // Static functions
        static STEPParser_CError      *SetConnectingSpaces(STEPParser __gc *parserObj);
                // Set up the connecting doors between spaces
        static STEPParser_CError      *CleanupGlobalList(STEPParser __gc *parserObj);
                // Remove excess doors from global list
```

Chunk 5:

This chuck of the header defines data for the class which may be derived from other IFC related classes such as STEPParser_IfcSpace and STEPParser_IfcBoundingBox in this case. Data elements are often declared as pointers which are then allocated in the class constructor. Some data elements are specified as being publicly available whilst others remain private to the class. The convention used in the code is to prefix private data with the underscore symbol. Finally the header is terminated with compiler-specific commands.

```
        // Data
        XmlElement                  *xmlElem;          // XML equivalent element
        STEPParser_coordStruct      placement;         // Absolute placement
        STEPParser_IfcSpace         *parentSpace;      // Parent space
        STEPParser_IfcBoundingBox *boundingBox;        // Bounding box that defines door
        ArrayList       *spaceConnections; // List of spaces that the door connects

 private:
        STEPParser_measure              _frameWidth;   // Width of door frame
        STEPParser_measure              _frameDepth;   // Depth of door frame
};

////////////////////////////////////////////////////////////////////////

#endif // !defined(AFX_STEPPARSER_IFCDOOR_HPP__191D52C6_B3F3_43A4_9263_503C667164A5
       __INCLUDED_)
```

11.12.4 An example `Parse` method

Each IFC entity processed by the **IfcSTEP Parser** has an attached "`Parse`" method. This method interrogates the entity and processes those components necessary for the translation. In many cases the `Parse` method for one entity will call `Parse` methods of sub-types as the **IfcSTEP Parser** program navigates the IFC Model tree. An example `Parse` method (again for the *IfcDoor* entity) is discussed below. The original source code is presented in chunks, with each chunk explained in detail.

Chunk 1:

The `Parse` method includes comments that provide the programmer with details of the required parameters and the return value. This is followed by a copy of the EXPRESS definition for the entity extracted from the IFC documentation.

```
///     P a r s e
///     =========
///
///     Extract any doors.
///
///     Parameters:
///             STEPParser *parseObj - parser object
///             IEntity *srcEntity - pointer to door STEP entity
///
///     Returns:
///             NO_ERROR
///
///  ENTITY IfcDoor;
///     ENTITY IfcRoot;
///             GlobalId    :   IfcGloballyUniqueId;
///             OwnerHistory    :   IfcOwnerHistory;
///             Name    :   OPTIONAL IfcLabel;
///             Description    :   OPTIONAL IfcText;
///     ENTITY IfcObject;
///             ObjectType    :   OPTIONAL IfcLabel;
///     INVERSE
///             IsDefinedBy    :   SET OF IfcRelDefines FOR RelatedObjects;
///             HasAssociations    :   SET OF IfcRelAssociates FOR RelatedObjects;
///             HasAssignments    :   SET OF IfcRelAssigns FOR RelatedObjects;
```

```
///            Decomposes   :   SET [0:1] OF IfcRelDecomposes FOR RelatedObjects;
///            IsDecomposedBy   :   SET OF IfcRelDecomposes FOR RelatingObject;
///    ENTITY IfcProduct;
///            ObjectPlacement   :   OPTIONAL IfcObjectPlacement;
///            Representation   :   OPTIONAL IfcProductRepresentation;
///    INVERSE
///            ReferencedBy   :   SET OF IfcRelAssignsToProduct FOR RelatingProduct;
///    ENTITY IfcElement;
///            Tag   :   OPTIONAL IfcIdentifier;
///    INVERSE
///            FillsVoids   :   SET [0:1] OF IfcRelFillsElement
///                    FOR RelatedBuildingElement;
///            ConnectedTo   :   SET OF IfcRelConnectsElements FOR RelatingElement;
///            HasCoverings : SET OF IfcRelCoversBldgElements
///                    FOR RelatingBuildingElement;
///            HasProjections   :   SET OF IfcRelProjectsElement FOR RelatingElement;
///            HasPorts   :   SET OF IfcRelConnectsPortToElement FOR RelatedElement;
///            HasOpenings : SET OF IfcRelVoidsElement FOR RelatingBuildingElement;
///            IsConnectionRealization : SET OF IfcRelConnectsWithRealizingElements
///                    FOR RealizingElements;
///            ProvidesBoundaries : SET OF IfcRelSpaceBoundary
///                    FOR RelatedBuildingElement;
///            ConnectedFrom   :   SET OF IfcRelConnectsElements FOR RelatedElement;
///            ContainedInStructure : SET [0:1] OF IfcRelContainedInSpatialStructure
///                    FOR RelatedElements;
///    ENTITY IfcBuildingElement;
///    ENTITY IfcDoor;
///            OverallHeight   :   OPTIONAL IfcPositiveLengthMeasure;
///            OverallWidth   :   OPTIONAL IfcPositiveLengthMeasure;
///  END_ENTITY;
///
```

Chunk 2:

This chunk defines the method and the variables. A standardised naming convention has been used in the **IfcSTEP Parser** program for the C++ classes where '`STEPParser_...`' defines a class written for the **IfcSTEP Parser** program. Class names beginning with '`I`', such as `IAttributesPtr` are defined by the **SECOM Server**.

The class returns a pointer to a `STEPParser_CError` structure so that errors can be communicated to other classes that call the `Parse` method. If no error occurs then the pointer to the `STEPParser_CError` is not set.

Variables that exist within the scope of the method are defined using the naming convention discussed above. The programmer can also enable additional debugging output by uncommenting the `//#define LOCALxDEBUG` line.

```
STEPParser_CError *STEPParser_IfcDoor::Parse(STEPParser __gc *parseObj,
   IEntity *srcEntity)
{
        //#define LOCALxDEBUG                   // Output debug info
        String                                  *val;
        IAttributesPtr                          attrs;
        IAttributePtr                           attr;
        IEntitiesPtr                            relatedObjects;
        IEntitiesPtr                            ifcRelAggregatesColl, ifcRelDefinesColl;
        IEntitiesPtr                            ifcRelAssociatesColl;
        IEntityPtr                              entity;
        IEntityPtr                              ifcRelAggregates, ifcRelDefines;
        IEntityPtr                              ifcRelAssociates;
        STEPParser_IfcObjectPlacement                   *objectPlacement;
        STEPParser_IfcProductRepresentation             *productRepresentation;
        STEPParser_IfcSpatialStructureElement           *spatialStructureElement;
        STEPParser_IfcPropertySetDefinition             *propertySetDefinition;
        STEPParser_IfcMaterialLayerSetUsage             *materialLayerSetUsage;
```

Chunk 3:

This chunk defines the output information and format sent to the log file. The STEP object identification reference and name are sent by default and the programmer can optionally enabled additional log details by using the `logFlag` variable. The use of the flags means that settings can be made to cascade down to sub-entites making control of the log file output more flexible. This chunk also illustrates where a portion of the **IfcSTEP Parser** program still runs in 'unmanaged' mode. The log file capability has yet to be updated and so runs as an 'unmanaged' class indicated by the programmer created `um->` pointer.

```
        // Log settings
        parseObj->um->log.Indent();
        logFlag = parseObj->um->log.GetFlag(THISxLOGxOBJECTxFLAG);
        parseObj->um->log.Log(logFlag, "IfcDoor (%s) : %s\n", this->SetID(srcEntity),
                this->SetName(srcEntity));
        logFlag = false;        // Disable logs
        parseObj->um->log.
                ChangeFlag(STEPParser_LOGxOBJECT_MATERIALxLAYERxSETxUSAGE,logFlag);
        parseObj->um->log.
                ChangeFlag(STEPParser_LOGxOBJECT_PROPERTYxSETxDEFINITION, logFlag);
        parseObj->um->log.
                ChangeFlag(STEPParser_LOGxOBJECT_OPENINGxELEMENT, logFlag);
        parseObj->um->log.
                ChangeFlag(STEPParser_LOGxOBJECT_OBJECTxPLACEMENT, logFlag);
        parseObj->um->log.
                ChangeFlag(STEPParser_LOGxOBJECT_PRODUCTxREPRESENTATION, logFlag);


        // Debug
        #ifdef LOCALxDEBUG
        debugParse(&(parseObj->um->log), srcEntity);
        #endif  //LOCALxDEBUG
```

Chunk 4:

This chunk sets up pointers to associated variables used during the processing of the entity. In this example the parent space of the door is linked to the space currently

being processed and the door object is added to the `global_doorList` for later identification of space connections.

```
// Define parent space of door as the current space being
// processed and add the door to the global door list.
this->parentSpace = parseObj->currentSpace;
parseObj->global_doorList->Add(this);
```

Chunk 5:

The chunk parses the associated *IfcProductRepresentation* entity specified by the Representation link. Function calls to the **SECOM Server** are used to extract the correct *IfcProductRepresentation* entity and the dimensions of the door object are identified. Chunks of code that deal with specific sections of the IFC Model structure have a comment section at the beginning that shows the associated EXPRESS documentation.

```
// Parse Product Representation
//     ENTITY IfcProduct;
//             Representation   :   OPTIONAL IfcProductRepresentation;
attrs = srcEntity->GetAttributes();
attr = attrs->Item(_variant_t("Representation"));
if (attr != NULL)
        {
        VERIFY(productRepresentation =
                new STEPParser_IfcProductRepresentation);
        productRepresentation->Parse(parseObj, entity);
        this->boundingBox = productRepresentation->
                shapeRepresentation->boundingBox;
        }
        else
        {
        // Set dimensions using overall dimensions
        GetAttribute(&val, srcEntity, "OverallHeight");
        this->boundingBox->zdim = Convert::ToInt32(val);
        GetAttribute(&val, srcEntity, "OverallWidth");
        this->boundingBox->xdim = Convert::ToInt32(val);
        }
```

Chunk 6:

Here the position of the entity is found using the *IfcObjectPlacement* entity. Since entities are placed relative to parent entities in the IFC Model, the parsing of the object placement tree can be quite complex if the current entity is nested deep in the tree. The parsing of the object placement tree is undertaken by iterative uses of the *IfcObjectPlacement* entity in the `STEPParser_IfcObjectPlacement` class (see Figure 11-18). On completion of the iterative process, the absolute position and facing direction of the entity is obtained and stored.

```
// Parse object placement
//      ENTITY IfcProduct;
//              ObjectPlacement   :   OPTIONAL IfcObjectPlacement;
attrs = srcEntity->GetAttributes();
attr = attrs->Item(_variant_t("ObjectPlacement"));
if (attr != NULL)
        {
        entity = attr->GetValue();
        if (entity != NULL)
                {
                // Clear temporary storage
                parseObj->ClearTmpPt();
                parseObj->ClearTmpDir();

                // Determine ObjectPlacement
                VERIFY(objectPlacement = new STEPParser_IfcObjectPlacement);
                objectPlacement->Parse(parseObj, entity);

                // Set coordinates
                this->placement.c[STEPParser_COORDxX] =
                        parseObj->tmpPt->c[STEPParser_COORDxX];
                this->placement.c[STEPParser_COORDxY] =
                        parseObj->tmpPt->c[STEPParser_COORDxY];
                this->placement.c[STEPParser_COORDxZ] =
                        parseObj->tmpPt->c[STEPParser_COORDxZ];

                // Debug
                #ifdef LOCALxDEBUG
                if (parseObj->um->log.GetFlag(THISxLOGxOBJECTxFLAG))
                        {
                        parseObj->um->log.Log(true, "Absolute placement
                                %.2f, %.2f, %.2f\n",
                                this->placement.c[STEPParser_COORDxX],
                                this->placement.c[STEPParser_COORDxY],
                                this->placement.c[STEPParser_COORDxZ]);
                        parseObj->um->log.Log(true, "Direction
                                %.2f, %.2f, %.2f\n",
                                parseObj->tmpDir->c[STEPParser_COORDxX],
                                parseObj->tmpDir->c[STEPParser_COORDxY],
                                parseObj->tmpDir->c[STEPParser_COORDxZ]);
                        }
                #endif
                }
        }
```

Chunk 7:

Any associated *IfcSpatialStructureElement* entities are identified in this chunk using the functionality of the **SECOM Server** classes. If any *IfcSpatialStructureElement* entities are found they are parsed by the `STEPParser_IfcSpatial-StructureElement` class.

```
// Determine RelatedElements from IfcRelContainedInSpatialStructure
//      ContainedInStructure : SET [0:1] OF IfcRelContainedInSpatialStructure
//    FOR RelatedElements;
ifcRelAggregatesColl = srcEntity
        ->GetUsedIn("IfcRelContainedInSpatialStructure", "RelatedElements");
if (ifcRelAggregatesColl != 0)
        {
        // Loop through RelAggregates collection
        for (long i = 1; i <= ifcRelAggregatesColl->GetCount(); i++)
                {
                // Get attributes of the RelAggregates collection
                ifcRelAggregates = ifcRelAggregatesColl->Item(_variant_t(i));
                attrs = ifcRelAggregates->GetAttributes();
```

```
                for (long j = 1; j <= attrs->GetCount(); j++)
                        {
                        attr = attrs->Item(_variant_t(j));
                        if (attr->GetType() ==
                                _bstr_t("IfcSpatialStructureElement"))
                                {
                                entity = attr->GetValue();
                                if (entity != NULL)
                                {
                                VERIFY(spatialStructureElement =
                                   new STEPParser_IfcSpatialStructureElement);

                                spatialStructureElement->Parse(parseObj, entity);
                                }
                                }
                        }
                }
        }
```

Chunk 8:

The chunk follows a similar procedure as shown for Chunk 7, identifying any associated *IfcPropertySetDefinition* and *IfcMaterialLayerSetUsage* entities. The entities define the properties of the door and the material layers that are used to construct the door. Parsing methods for other IFC Model entities may have chunks similar to this depending on the particular structure of the entity.

```
// Find IfcPropertySetDefinition entities
//      ENTITY IfcObject;
//      INVERSE
//              IsDefinedBy     :   SET OF IfcRelDefines FOR RelatedObjects;
ifcRelDefinesColl = srcEntity->GetUsedIn("IfcRelDefines", "RelatedObjects");
if (ifcRelDefinesColl != NULL)
        {
        // Loop through Rel collection
        for (long i = 1; i <= ifcRelDefinesColl->GetCount(); i++)
                {
                // Get attributes of the Rel collection
                ifcRelDefines = ifcRelDefinesColl->Item(_variant_t(i));
                attrs = ifcRelDefines->GetAttributes();
                for (long j = 1; j <= attrs->GetCount(); j++)
                        {
                        attr = attrs->Item(_variant_t(j));
                        if (attr->GetType() ==
                                _bstr_t("IfcPropertySetDefinition"))
                                {
                                entity = attr->GetValue();
                                if (entity != NULL)
                                {
                                // Parse PropertySetDefinition
                                VERIFY(propertySetDefinition = new
                                        STEPParser_IfcPropertySetDefinition);
                                propertySetDefinition->Parse(parseObj, entity);
                                }
                                }
                        }
                }
        }


// Determine RelatedObjects from IfcRelAssociates...
//              HasAssociations     :   SET OF IfcRelAssociates FOR RelatedObjects
ifcRelAssociatesColl = srcEntity->
        GetUsedIn("IfcRelAssociates", "RelatedObjects");
if (ifcRelAssociatesColl != NULL)
```

```
{
// Loop through RelAssociates collection
for (long i = 1; i <= ifcRelAssociatesColl->GetCount(); i++)
        {
        // Get attributes of the RelAssociates collection
        ifcRelAssociates = ifcRelAssociatesColl->Item(_variant_t(i));
        attrs = ifcRelAssociates->GetAttributes();
        for (long j = 1; j <= attrs->GetCount(); j++)
                {
                // Check for IfcMaterialSelect entity
                attr = attrs->Item(_variant_t(j));
                if (attr->GetType() == _bstr_t("IfcMaterialSelect"))
                        {
                        // Determine if material/s are selected through
                        // IfcMaterial, IfcMaterialList,
                        //  IfcMaterialLayer, IfcMaterialLayerSet or
                        // IfcMaterialLayerSetUsage entity
                        entity = attr->GetValue();
                        if (entity != NULL)
                        {
                        if (entity->GetType() ==
                                _bstr_t("IfcMaterialLayerSetUsage"))
                                {
                                VERIFY(materialLayerSetUsage = new
                                    STEPParser_IfcMaterialLayerSetUsage);
                                materialLayerSetUsage->Parse(parseObj,
                                    entity);
                                }
                        }
                        }
                }
        }
}
```

Chunk 9:

Finally the Parse method is terminated, returning a NO_ERROR value to indicate a successful outcome.

```
        parseObj->um->log.Indent(-1);
        return NO_ERROR;
        #undef LOCALxDEBUG
}
```

11.13 The IfcSTEP-BRANZFIRE Parser program

The '**BRANZFIRE**' module consists of 3000 lines of C++ code which interfaces with the core **IfcSTEP Parser** program code. The process for creating **BRANZFIRE** input files is illustrated in Figure 11-9 and can be broken into three main phases. Phase one is to process IFC file into intermediate data structures using the core **IfcSTEP Parser** program similar to the 'vanilla' version of the **IfcSTEP Parser** program. Phase two processes the intermediate data structures to create **BRANZFIRE** input file using the **IfcSTEP-BRANZFIRE Parser** program code. Finally phase three writes the **BRANZFIRE** input file to disk using a set of generic **BRANZFIRE** classes.

As with the core **IfcSTEP Parser** program code, it is not practical to include a full listing of the **IfcSTEP-BRANZFIRE Parser** program code and the generic **BRANZFIRE** classes. The generic **BRANZFIRE** classes include methods and data that process rooms, objects (i.e. burning items) and vents plus data structures for material and boundary definitions relevant to **BRANZFIRE** input files as illustrated in Figure 11-13.

Chapter 12:
CONCLUSIONS AND RECOMMENDATIONS

12.1 Summary

This thesis demonstrates that the exchange of fire engineering related electronic information is feasible and practical using emerging industry standard technologies. In particular, the ability to transfer IFC Model compliant files generated by a commercially available CAD package into a commonly used fire simulation tools has been demonstrated. In addition the thesis has created a schema for the creation of online databases of rate of heat release information which can be incorporated into the transfer process. The thesis has identified a number of key issues regarding the integration of online databases and building product models with fire simulation software and these are summarised below. Although much has been achieved through the work that makes up this thesis, considerable limitations have been identified. Thus the process of exchange of fire engineering related electronic information is far from complete and suggestions for future work are presented.

12.2 Online databases

This thesis has created a schema for online databases of fire rate of heat release information using XML technology (Chapter 5). The use of the XML technology means that the database schema can be upgraded to include additional information without compromising the functionality of software tools developed before enhancement. It has been shown in Chapter 6 that databases using the schema can be accessed through web pages or client software including the **BRANZFIRE** fire simulation software tool. Database records can be transformed into a variety of formats that can be read by a range of generic and fire engineering specific applications. Finally database records can also be integrated with the IFC Model's 'property set definition' requirements. Online databases of other fire engineering related properties such as for fire protection system components are the subject for further research outside the scope of this thesis.

12.3 Content of the IFC Model

The IFC Model is ideally suited to meet the requirements set out by Mowrer and Williamson (1988). The object-oriented structure of the Model and the ability to associate properties to objects were two of the key points identified by Mowrer and Williamson. The fact that the IFC Model is designed to be an open standard that uses

internationally agreed technologies, that it is being used by a diverse range of domains within the construction industry and that it is implemented by several commercially available CAD packages, all enhance the position of the IFC Model. As discussed in Chapter 4, there are potential benefits for the fire engineering community to use the IFC Model.

This study has examined the content of the IFC Model with particular reference to its fire engineering content. Although, as discussed in Chapter 7, earlier versions of the IFC Model only had a limited set of fire engineering related properties, the current IFC 2x2 release has a considerable amount of material that is potentially useful to fire engineers. It has been found that there are still opportunities to make revisions to the IFC Model where fire engineering is concerned and reviews of the Model, such as that presented in Chapter 9, are of benefit to the IFC Model developers.

The IFC Model is limited in that it is not able to define properties for every building element that may exist. The IFC Model's property set definition mechanism overcomes this limitation by allowing extensions to be made to the IFC Model. As discussed in Chapter 6, the property set definition approach was used to enable rate of heat release data to be added to furniture items. As the IFC Model matures it is expected that further enhancements will be made that will add new entities and property sets as well as refining those that already exist in the current version.

12.4 Implementation of the IFC Model in CAD packages

This thesis identified in Chapter 8 and Chapter 11 two commercially available CAD packages that were able to exchange the IFC Model, although the versions of the IFC Model and the file output format implemented by the CAD packages differed. However, the user's ability to exchange information is constrained where a CAD vendor does not keep up with the most recently released versions of the IFC Model.

It was also found that these CAD packages did not always implement every facet of the IFC Model. This resulted in some difficulties in creating and populating some of the IFC entities. There is also an associated complexity involved with the creation of

large buildings in powerful CAD packages where the exchange process is limited by the ability of the user to make use of the sophisticated software.

Thus the mere existence of an entity or property in the IFC Model does not guarantee that a user will be able to easily create it in their chosen CAD package. This limitation will have potential knock-on effects on the availability of IFC entities and their mapping to fire simulation software.

12.5 Extraction of elements from an IFC file

The thesis has demonstrated that there are limitations on the ability to extract elements from an IFC file. These restrictions occur as a result of the work required to write algorithms to extract every IFC entity and the ability to interrogate IFC files that can be delivered in various formats. The availability of software such as the **MSXML** and **SECOM Server** assist greatly with the interrogation process as they relieve the developer of the considerable amount of effort in writing their own interrogation algorithms. The IFC Parsers written for this thesis only process the limited set of the IFC Model required to obtain basic building geometry and properties. The thesis has noted that it is unlikely that every IFC entity will be applicable to all domains and this means that it would be unnecessary for algorithms to be written for all of the existing entities. The study has shown where further work is required to expand the scope of IFC entities that can be processed and the ability to use either STEP or XML file encodings.

12.6 Mapping the IFC Model to fire simulation software

As discussed in Chapter 10, the mapping from the IFC Model to fire simulation software is constrained by the representation of the fire scenario as well as the implementation of that representation in a specific fire simulation software tool. The zone modelling approach to simulating the fire scenario has particular requirements and restrictions regardless of the software tool it is implemented in. However a specific software tool may also place further constraints on the ability to map IFC entities.

The study has also shown where further work is required to improve the mapping of the IFC entities to the requirements of zone fire simulation software and enlarging the range of fire simulation software that can interface to the software. However it is important to recognise that the requirements for a structural response analysis tool will be quite different to those for a people movement simulation, for example, and considerable effort is likely to be required to identify appropriate mappings for each type of software tool. As with zone modelling software, the modelling technique and the software implementation of that technique may impose constraints on the mapping process.

12.7 Consistent terminology

Since the IFC Model is designed to be a general product model for buildings then definitions used by the authors of the IFC Model need to be sufficiently familiar across the wide range of domains involved within the construction industry. These IFC definitions may vary from those commonly used within the fire engineering domain as illustrated in Chapter 11 where the IFC Model includes *IfcSpace* and *IfcZone* entities compared with **BRANZFIRE**'s use of rooms and the often used fire engineering term 'compartment'.

Throughout this thesis it has been seen that it is essential that terminology be clearly defined in order to facilitate the exchange process between the IFC Model and various fire simulation software tools. Without a clear understanding of what terms are being used to describe specific building elements and properties it becomes difficult to create direct mapping across software tools. This difficulty is likely to be exacerbated when considering terminology across domains. By highlighting such variations it is hoped that fire engineering and IFC terminology may gradually merge.

12.8 Benefits to software users

Chapter 8 and Chapter 11 show that it is feasible to use the IFC Model as a means of exchanging building representations in an electronic form. However, as already summarised above, this exchange does not come without its challenges and Section 11.6.1 discusses whether this results in an overall benefit to users. Clearly, there is a significant amount of future work that should be considered before there is

seamless data exchange amongst the various software tools used by fire engineers. This thesis has pointed to where some of that work should be directed.

12.9 Recommendations for future work

In order to provide further momentum to establishing electronic exchange technologies within the fire engineering community it would seem appropriate to form an international group, organised by a professional body such as the Society of Protection Engineers (SFPE). The group could provide input to the development of the IFC Model from a fire engineering perspective as well as disseminating information back to the fire engineering community. Such a group would need to include representatives from fire protection equipment manufacturers, fire simulation software developers, regulatory and standardisation bodies, professional fire engineers, research and training establishments as well as specialist IT people.

The inclusion of manufacturers, or trade groups representing particular sectors of the fire protection industry, would enable the development of industry agreed specifications for online catalogues of fire protection product data. These specifications would need to identify the parameters that are applicable to each item or class of fire protection product. Parameters could be mandatory or optional and be openly published or confidential to a selected group. Manufacturers could then open up their product databases to designers so that relevant information can be easily transferred into an electronic design tool such as a CAD package. Similarly, fire testing bodies and research establishments could provide online databases of product assessments either directly or through the product manufacturer.

Fire simulation software developers could then create interfaces to the IFC Model and the online catalogues for their particular simulation tools. Developers could update currently available fire simulation software to read IFC files either by writing interface tools that can operate separately from the target simulation software or by incorporating the translation process into the simulation package. Potential simulation tools include the National Institute of Standards and Technology (NIST) Fire Dynamics Simulator (FDS) computational fluid dynamics code (McGrattan et al., 2002) which is becoming more widely used by fire engineers, the EvacuatioNZ

human movement simulation model (Ko et al., 2004) which was specifically written to use XML to define its building geometry information or many of the other fire simulation tools identified by Olenick and Carpenter (2003). NIST is already starting to examine links between IFC-based BIMs with their CONTAM (indoor contaminants), CFAST (zone-based fire and smoke transport) and FDS (CFD-based fire and smoke transport) simulation tools (Reed, 2005).

Regulators could provide fire code documents in a form suitable for incorporation into various electronic systems. Regulators could also enable the wider spread use of electronic exchange by requiring certain practices be adopted such as the use of the IFC Model. For example, it is reported (Reed, 2005) that the General Services Administration in the US will begin requiring the submission of building information model data in IFC format as early as 2006. Similarly, the Singapore Government's CORENET initiative (Building and Construction Authority, 2004) aims to use IFC to deliver a single plan-checking tool to assess building plans submitted by designers for compliance of the various authorities regulations. In addition a project led by NCS/BCS (National Conference of States on Building Codes and Standards) with NIST as a partner is reported to have looked at the use of IFC-based BIMs in submission and review of designs for the New York City Department of Buildings (AIA, 2005). Such requirements will encourage designers to use IFC compliant software which in turn will provide a market for CAD vendors to sell IFC compliant packages.

The fire engineering professionals would represent the interests and requirements of a large proportion of the end users of fire simulation software and similar tools. They would see the potential benefits of electronic exchange technologies and drive demand for their adoption through the fire engineering community and somewhat to the wider architectural and engineering community.

Educators would be involved in disseminating electronic exchange technologies to current and upcoming practitioners. Education would involve getting practitioners to understand and use electronic exchange methods as part of their design process and providing skills to allow for new developments to be made through research projects. The construction information technology content of most undergraduate civil

engineering courses has been found to be necessarily limited and educational programmes that address these topics are already under development (Rebolj et al., 2004). This author is not aware of any courses that specifically cover construction information technology in the main international fire engineering educational programmes over and above student's exposure to fire simulation tools and an expectation that they can use general computer packages such as word processors, spreadsheets, simple 2D CAD etc.

Finally, specialist IT people would be able to support the above tasks through their intimate knowledge of database systems, building information modelling tools, electronic exchange technologies, programming languages etc. It is not possible to expect the fire engineering community to understand all of these aspects at a sufficient level of detail to be able to manage without the input of IT specialists.

Chapter 13:
REFERENCES

Adachi Y (2002). PSD Worksheet reference. International Alliance for Interoperability, MSG-01-2002, Draft 1.

Adachi Y, Forester J, Hyvarinen J, Karstila K, Liebich T, Wix J (2003). Industry Foundation Classes IFC2x Edition 2, Modeling Support Group, International Alliance for Interoperability.

aecXML Working Group (2001). A framework for electronic communications for the AEC industries. http://www.aecxml.org/docs/aecwhite.doc (accessed May 2001).

AIA (2005). National Institute of Standards and Technology - Data Exchange Standards Activity. American Institute of Architects Technology in Architectural Practice Knowledge Community http://www.building-connections.info/organizations/nist.htm, (accessed November 2005).

Amor R, Jain S, Augenbroe G (2004). Online product libraries: The state-of-the-art. Proceedings of the CIB Triennial, Toronto, Canada, 3 - 7 May.

Anon (2003). Beyond CAD. CAD User AEC Magazine, Vol. 16, No. 07 - July/Aug 2003, http://www.caduser.com/reviews/reviews.asp?a_id=157, (accessed Dec 2004).

Anon (2005). Data modelling using EXPRESS-G for IFC development. http://www.iai-international.org/iai_international/Technical_Documents/documentation/Data_Modelling_Using_EXPRESS-G_for_IFC_Development.pdf (accessed January 2005).

Augenbroe G L M (1994). An overview of the COMBINE project. Proceedings of the First ECPPM Conference, Dresden.

Babrauskas V, Peacock R D (1992). Heat release rate: The single most important variable in fire hazard. Fire Safety Journal. 18 (3) 255-272.

Bazjanac V, Maile T (2004). IFC HVAC interface to EnergyPlus – a case of expanded interoperability for energy simulation. SimBuild 2004, IBPSA-USA National Conference, Boulder, CO, August 4 – 6.

Berkhahn V, Esch C (2003). Re-engineering of objects in construction drawings. CIB W78 20th International Conference on Information Technology in Construction, Auckland, New Zealand.

BLIS Project Companies (2002). Building Lifecycle Interoperable Software - Project Brief. http://www.blis-project.org (Accessed December 2002 and March 2003).

Bloomfield D (1994). IT in the construction industry. BRE Client Report CR68/94, Building Research Establishment, Garston, Watford.

BIA (2001). Approved Document for New Zealand Building Code, Fire Safety Clauses. Acceptable Solution C/AS1. Building Industry Authority, Wellington.

British Standards Institution (1995). Fire detection and alarm systems for buildings, Part 6: Code of practice for the design and installation of fire detection and alarm systems in dwellings. BS 5839: Part 6: 1995.

Building and Construction Authority (2004). CORENET (COnstruction and Real Estate NETwork), http://www.corenet.gov.sg/ (accessed April 2005).

Bukowski R W, Peacock R D , Jones W W, Forney C L (1989). Software user's guide for the HAZARD I fire hazard assessment method. NIST Handbook 146, Volume I. Gaithersburg, National Institute of Standards and Technology.

Bukowski R (1996). Applications of FASTLite. Proc. Computer Applications in Fire Protection Engineering, Technical Symposium, June 20-21, 1996, Worcester, MA.

Caldas C H, Soibelman L, Han J (2002). Automated classification of construction project documents. Journal of Computing in Civil Engineering, pp.234-243.

Cassidy M A (1999). An integrated database for the construction industry, The Construction Specifier, http://old.csinet.org/xp/p-cs/i-1999062301/a-930921174/article.view (accessed November 2004).

Cheng F (1995). BRE engineering data model – BREED 2.3.0. BRE Note N95/95. Building Research Establishment, Garston, Watford.

Chitty R, Cox G (1988). ASKFRS: An interactive computer program for conducting fire engineering estimations. Building Research Establishment, Borehamwood.

Chitty R, Spearpoint M J (1996). JROOM: An interface to JASMINE. BRE Note N71/96, Building Research Establishment, Borehamwood.

Cooper L Y, Stroup D W (1982) Calculating Available Safe Egress Time (ASET) - A computer program and user's guide, NBSIR 82-2578, National Bureau of Standards, Washington DC.

Deal S (1995). Technical reference guide for FPEtool version 3.2. NISTIR 5486-1. Gaithersburg, National Institute of Standards and Technology, MD.

Finch E (2000). Net gain in construction: Using the Internet in the construction industry. Butterworth-Heinemann, Oxford, ISBN 0750650524.

Fraser-Mitchell J (1994). An object-oriented simulation (CRISP II) for fire risk assessment. Proceedings of the Fourth International Symposium on Fire Safety Science, July 13-17, 1994, Ottawa, Ontario, Canada, Intl. Assoc. for Fire Safety Science, Boston, MA, Kashiwagi, T., Editor, pp.793-804.

Frost I, Patel M K, Galea E R, Rymacrzyk P, Mawhinney R N (2001). A semi-automated approach to CAD input into filed based fire modelling tools. Proc. Interflam 2001, Edinburgh, Scotland, pp.1421-1426.

Goodman D (2001). JavaScript Bible (4th ed.). Hungary Minds Inc., Indianapolis, USA, ISBN 0-7645-3342-8.

Graphisoft (2001). Industrial Use of the Building Modelling Approach. interop AEC+fm 2001, Sydney, Australia 2001.

Graphisoft (2003). ArchiCAD 8 user guide.

Harold E R (1998). XML: Extensible Markup Language. IDG Books Worldwide, Inc., Foster City, CA, USA. ISBN 0-7645-3199-9.

Howe D (ed.) (1993). The Free On-line Dictionary of Computing, http://www.foldoc.org/ (accessed November 2004 and January 2005).

Ito K (1995). General product model and domain specific product model in the A/E/C industry. Proc. 2nd Congress on Computing in Civil Engineering, Atlanta, GA. Vol. 1, pp.13-16.

IAI (2004a). International Alliance for Interoperability, http://www.iai-international.org/iai_international/ (accessed November 2004).

IAI (2004b). IFC Model development extension projects - AR-4 Escape Route Planning, International Alliance for Interoperability, http://ce.vtt.fi/iaiIFCprojects/ (accessed January 2004).

IAI (2004c). IFC2x Edition 2 Addendum 1 specification, Model Support Group (MSG) of the IAI, International Alliance for Interoperability.

ISO (1988). ISO 8879:1986 (Amendment 1:1988), Information processing - Text and office systems - Standard Generalized Markup Language (SGML), International Organization for Standardization.

ISO (1993). ISO 9705:1993. Fire tests on building materials and structures - Part 33. Full-scale room test for surface products, International Organization for Standardization.

ISO (1994). ISO 10303-11:1994. Industrial automation systems and integration - Product data representation and exchange - Part 11: Description methods: The EXPRESS language reference manual, International Organization for Standardization.

ISO (1999). ISO 834-1, Fire-resistance tests - Elements of building construction. International Organization for Standardization.

ISO (2002). ISO 10303-21:2002 Industrial automation systems and integration - Product data representation and exchange - Part 21: Implementation methods: Clear text encoding of the exchange structure. International Organization for Standardization.

Jason N H (1996). Locating fire information. Interflam '96 – Proc. 7th International Interflam Conference, Interscience Communications Ltd., London, England, pp. 691-698.

Jones W W, Forney G P, Peacock R D, Reneke P A (2000). A technical reference for CFAST: An engineering tool for estimating fire and smoke transport, NIST TN 1431, National Institute of Standards and Technology, Gaithersburg, MD.

Jung Y, Gibson G E (1999). Planning for computer integrated construction. Journal of Computing in Civil Engineering, October 1999, pp.217-225.

Karola A, Lahtela H, Hänninen R, Hitchcock R, Chen Q, Dajka S, Hagström K (2002). BSPro COM-Server – interoperability between software tools using industrial foundation classes. Energy and Buildings 34 pp.901-907.

Kim L (2003). The official XMLSpy handbook. Wiley Publishing, Indianapolis, ISBN 0-7645-4964-2.

King B J, Norman P W (1992). A Step in the right direction. Professional Engineering, November.

Ko S Y, Spearpoint M J, Teo A (2004). Trial evacuation of an industrial premises and evacuation model comparison. Submitted to Fire Safety Journal.

LandXML (2001). LandXML schema documentation (version 0.88). http://www.landxml.org/ (accessed May 2001).

Lewis J O, Kenny P (1994). COMBINE 2, Computer models for the building industry in Europe: Phase 2. The European Commission, Directorate-General XII for Science Research and Development.

Liebich T (2000). IFC support for XML. Working with XML in construction. IAI International Council and International Technical Management Meetings, London.

Liebich T, Wix J (2000). IFC technical guide, Industry Foundation Classes – Release 2x, International Alliance for Interoperability.

Liebich T (2001). XML schema language binding of EXPRESS for ifcXML. MSG-01-001(Rev 4), International Alliance for Interoperability.

Liebich T (ed.) (2003). IFC 2x Edition 2 model implementation guide. Version 1.6. Modeling Support Group, International Alliance for Interoperability.

Liebich T (ed.) (2004). IFC 2x Edition 2 model implementation guide. Version 1.7. Modeling Support Group, International Alliance for Interoperability.

Lietaer B (2001). The future of money. Century, London. ISBN 0 7126 8399 2.

Light R (1997). Presenting XML. Sams.net, Indianapolis, IN. ISBN 1-57521-334-6.

Lønvik L E, Opstad K (1992). DCS – Data converting system. SINTEF, Norway.

Lundin J (2001). Fire safety engineering resources on the internet. 2001-08-28 /JL, Dept. of Fire Safety Engineering, Lund University, Sweden, http://www.brand.lth.se/links/internet.pdf (accessed November 2004).

Massa R J, Cappuccio J A (1995). Architectural CAD-based, topological building models for fire hazard analysis. Proc. International Conference on Fire Research and Engineering (ICFRE). 10-15 Sept 1995, Orlando, FL. Lund D P, Angell E A (ed.), SFPE, Boston, MA, pp.327-331.

McGrattan K, Forney G P, Floyd J E, Hostikka S, Prasad K (2002). Fire Dynamics Simulator (Version 3) – User's guide. NISTIR 6784, 2002 ed., National Institute of Standards and Technology, Gaithersburg, MD.

Mowrer F W (1987). Microcomputer-based building fire safety analysis. Doctoral dissertation in fire protection engineering and combustion science, University of California, Berkeley.

Mowrer F W, Williamson R B (1988). Room fire modeling within a computer-aided design framework. International Association for Fire Safety Science. 2nd International Symposium. June 13-17, Tokyo, Japan, pp.453-462.

NFPA (1996). National Fire Alarm Code. NFPA 72, National Fire Protection Association, Quincy, MA.

Nell J (2001). STEP on a page, http://www.nist.gov/sc5/soap/ (updated 2001-November-28).

NIST (2001). Fires on the web. http://fire.nist.gov/fire/fires/fires.html (accessed May 2001).

Olenick S M, Carpenter D J (2003). An updated international survey of computer models for fire and smoke. Journal of Fire Protection Engineering, 15 (2) pp.87-110.

Open Design Alliance (2003). Why isn't DXF good enough? http://www.opendesign.com/about/whtpaper/whynot.htm (accessed October 2005).

Osterrieder P, Richter S, Fischer M (2004). A product data model for design and fabrication of timber buildings. Proc. World Conference on Timber Engineering, Lahti, Finland, Vol 1, pp 47-52.

Parry R, Wade C A, Spearpoint M J (2003). Implementing a glass fracture module in the BRANZFIRE zone model. Journal of Fire Protection Engineering, Vol. 13 No. 3, pp.157-183.

Portier R W, Peacock R D, Reneke P A (1997). Data structures for the Fire Data Management System, FDMS 2.0. NISTIR 6088, National Institute of Standards and Technology, Gaithersburg, MD.

Quintiere J G (2002). Compartment fire modeling. Chapter 3-5, The SFPE Handbook of Fire Protection Engineering (3rd ed.), pp.3-162 – 3-170.

Rebolj D, Menzel K (2004). Another step towards a virtual university in construction IT. ITcon, Vol. 9, http://www.itcon.org/2004/17/.

Reed K (2005). Standard building information models. BFRL Project information. http://www2.bfrl.nist.gov/projects/projcontain.asp?cc=8635015000 (accessed April 2005).

Rönneblad A (2003). Product models for concrete structures. Licentiate Thesis 2003:22, ISSN: 1402 - 1757, Luleå University of Technology, Sweden.

Särdqvist S (1993). Initial fires – RHR, smoke production and CO generation from single items and room fire tests. ISRN LUTVDG/TVBB-3070-SE, Lund University, Sweden.

SECOM Co., Ltd (2005). IFC SECOM Server. http://groups.yahoo.com/group/ifcsvr-users/IFCsvr300/ (Accessed January 2005).

Simon R, Schmidt M (2002). Teach yourself Visual C++ .NET, Sams Publishing, Indianapolis, USA, ISBN 0-672-32323-0.

Spearpoint M J (1992a). Fire modelling: Integration or communication. Private communication, Borehamwood, BRE.

Spearpoint M J (1992b) Initial study of computer fire models suitable for field investigations. BRE Note N122/92, Building Research Establishment, Garston, Watford.

Spearpoint M J (1992c). Computer modelling following the investigation of a fire in Frampton Court, Acton, April 1992. BRE Note N101/92, Building Research Establishment, Borehamwood.

Spearpoint M J (1992d). Computer modelling following the investigation of a fire in Charlecote Tower, Edgbaston, July 1992. BRE Client Report CR146/92, Building Research Establishment, Borehamwood.

Spearpoint M J (1993a). A collection of rate of heat release data suitable for computer fire models, BRE Note N13/93, Building Research Establishment, Garston, Watford.

Spearpoint M J (1993b). Computer modelling following a fire in a dwelling, Hendon, Sunderland, March 1993. BRE Note N106/93, Building Research Establishment, Borehamwood.

Spearpoint M J, Smithies J N (1993). An initial study of an object-oriented approach to fire risk assessment using a Virtual Reality system. BRE Note N115/93, Building Research Establishment, Borehamwood.

Spearpoint M J (1994a). Initial development of a fire risk assessment model using an object-oriented virtual reality system. BRE Note N40/94, Building Research Establishment, Borehamwood.

Spearpoint M J (1994b). An object-oriented version of the 'ASET-B' fire model for a Virtual Reality environment. Building Research Establishment, Borehamwood, (unpublished).

Spearpoint M J (1995a). CAD environment for fire risk assessment: review of development and interaction with the COMBINE programmes. BRE Client Report CR81/95, Building Research Establishment, Borehamwood.

Spearpoint M J (1995b). The conversion of STEP data into the Superscape virtual reality system. COMBINE II meeting, Garston, BRE, (unpublished).

Spearpoint M J (1995c). Superscape interface to BREED. BRE Note N120/95, Building Research Establishment, Borehamwood.

Spearpoint M J (1995d). Integrated predictive modelling: the conversion of STEP data files into the HAZARD I fire model. BRE Chief Executives seminar, Building Research Establishment, Garston, Watford, (unpublished).

Spearpoint M J (1996). Superscape interface to BREED: Further development. BRE Note N70/96, Building Research Establishment, Borehamwood.

Spearpoint M J, Shipp M P (1997). Virtual reality for the channel tunnel fire investigation. Fire Safety Engineering, 4 (6) pp.6-7.

Spearpoint M J (1998). A comparison of LES3D predictions with experimental data. Private communication, University of Maryland, College Park, USA.

Spearpoint M J, Smithies J N (1999). Practical comparison of smoke detector sensitivity standards. Proc. 11th International Conference on Fire Detection, AUBE '99 conference, Duisburg, Germany, pp.576-587.

Spearpoint M J, Mowrer F W, McGrattan K (1999). Simulation of a single compartment flashover fire using hand calculations, zone models and a field model. Third International Conference on Fire Research and Engineering. Chicago, pp.3-14.

Spearpoint M J (2001). The development of a web-based database of rate of heat release measurements using a mark-up language. 5th Asia-Oceania Symposium on Fire & Technology, Newcastle, Australia, pp.205-218.

Spearpoint M J (2003a). The potential impact of building product models on fire protection engineering. Fire Protection Engineering, Issue 19, pp.42-48.

Spearpoint M J (2003b). Properties for fire engineering design in New Zealand and the IFC building product model. CIB W78 Proc. 20th International Conference on Information Technology in Construction, Auckland, New Zealand. CIB Publication 284, ISBN 0-908689-71-3, pp.333-340.

Spearpoint M J (2003c). Integrating the IFC building product model with zone fire simulation software. Proc. International Conference on Building Fire Safety, QUT, Gardens Point Campus, Brisbane, Australia, pp. 56-66.

Spearpoint M J (2005a). Fire engineering properties in the IFC building product model and mapping to BRANZFIRE. Accepted for publication in the International Journal on Engineering Performance-Based Fire Codes.

Spearpoint M J (2005b). Transfer of architectural data from the IFC model to a fire simulation software tool. Submitted to Journal of Fire Protection Engineering.

Standards Australia (1989). AS 1603 Automatic fire detection and alarm systems – Part 2: Point type smoke detectors.

Standards New Zealand (1999). AS/NZS 1530.3, Simultaneous determination of ignitability, flame propagation, heat release and smoke release.

Thompson P A, Marchant E W (1995). A computer model for the evacuation of large building populations, Fire Safety Journal, Vol. 24, pp.131-148.

Ungerer M (2003). Taking STEP further. ISO Bulletin May 2003. http://www.iso.ch/iso/en/commcentre/isobulletin/articles/2003/pdf/prostep03-05.pdf (Accessed February 2005).

Wade C A (1999). BRANZFIRE: Engineering software for evaluating hazard of room lining materials. Proc. 8th International Interflam Conference, Volume 2. Interscience Communications Ltd., London, England, pp.1147-1152, 1999.

Wade C A (2002). BRANZFIRE Technical reference guide. Study Report No. 92, ISSN: 0113-367, BRANZ, Porirua City.

Wade C A (2003a). BRANZFIRE Technical Reference Guide. BRANZ Study Report 92 (revised). Building Research Association of New Zealand. Judgeford, Porirua City, New Zealand.

Wade C A (2003b). A user's guide to BRANZFIRE 2003. Building Research Association of New Zealand. Judgeford, Porirua City, New Zealand.

WordNet (1997). Princeton University, version 1.6, Lexico Publishing Group, LLC. Accessed from www.dictionary.com.

W3C (2000). Canonical XML, Version 1.0, http://www.w3.org/TR/2000/WD-xml-c14n-20001011, World Wide Web Consortium (accessed November 2004).

W3C (2001). XML Schema, http://www.w3.org/XML/Schema, World Wide Web Consortium (accessed May 2001).

Yabuki N, Shitani T (2003). An IFC-based product model for RC or PC slab bridges. Proc. 20th International Conference on Information Technology in Construction, CIB Publication 284, ISBN 0-908689-71-3.

APPENDIX A. PARAMETERS REQUIRED BY SELECTED FIRE SIMULATION SOFTWARE TOOLS

A.1 Parameter list

A.1.1 Environmental conditions

CFAST: Internal temperature (°C)
Internal pressure (Pa)
Internal station elevation (m)
External temperature (°C)
External pressure (Pa)
External station elevation (m)
Wind speed (m/s) and associated factors to determine wind speed as a function of elevation

A.1.2 Space geometry

FPEtool: Room ceiling height (m)
Room length (m)
Room width (m)
Height of opening (m)
Width of opening (m)
Height of opening sill above floor (m)

CFAST: Compartment width (m)
Compartment depth (m)
Compartment height (m)
Floor elevation with respect to internal station elevation (m)
Opening width (m)
Opening sill height with respect to internal station elevation (m)
Opening soffit height with respect to internal station elevation (m)
Opening facing with respect to wind direction

A.1.3 Space topology

CFAST: Opening connections between two compartments

A.1.4 Space boundary

FPEtool: Number of materials and percentage comprised of material if more than one material

A.1.5 Space boundary materials

FPEtool: Thermal conductivity (kW/m^{2}K)
Density (kg/m^3)
Specific heat (kJ/kg K)
Thermal inertia (kW^{2}s/m4K^2)
Material thickness (mm)

CFAST: Volume density (kg/m^3)
Areal density (kg/m^2)
Conductivity (kW/m K)
Specific heat (kJ/kg K)
Emissivity (fraction)

A.1.6 Fire safety equipment

Heat detector

FPEtool: Radial distance from fire (m)
Head response time index ([m/s]1/2)
Head rating (°C)
In sidewall location (Y/N)

Sprinkler

FPEtool: Radial distance from fire (m)
Head response time index ([m/s]1/2)
Head rating (°C)
In sidewall location (Y/N)

Smoke detector

FPEtool: Radial distance from fire (m)
Smoke temperature at detection (°C)
In sidewall location (Y/N)

A.1.7 Mechanical HVAC system

FPEtool: Ventilation rate (air changes/hour)

APPENDIX B. FIREBASEXML SCHEMA

B.1 Complete schema listing

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.2 U (http://www.xmlspy.com) by Michael Spearpoint (University of Canterbury) -->
<!--W3C Schema generated by XML Spy v3.5 (http://www.xmlspy.com)-->
<!--
*       FireBase version 0.42, 27-Aug-2002, (c) Michael Spearpoint
-->
<?xml-stylesheet type="text/xsl" href="viewSchema_FireBaseXML.xslt"?>
<xsd:schema xmlns:xsd="http://www.w3.org/2001/XMLSchema" elementFormDefault="qualified">
    <xsd:complexType name="attributeType">
        <xsd:attribute name="Chair" type="xsd:string"/>
        <xsd:attribute name="Refuse" type="xsd:string"/>
        <xsd:attribute name="Bedding" type="xsd:string"/>
        <xsd:attribute name="Furniture" type="xsd:string"/>
        <xsd:attribute name="Vehicle" type="xsd:string"/>
        <xsd:attribute name="Cooking" type="xsd:string"/>
        <xsd:attribute name="Lining" type="xsd:string"/>
        <xsd:attribute name="Armchair" type="xsd:string"/>
        <xsd:attribute name="Loveseat" type="xsd:string"/>
        <xsd:attribute name="Pallet" type="xsd:string"/>
        <xsd:attribute name="Sofa" type="xsd:string"/>
        <xsd:attribute name="Appliance" type="xsd:string"/>
        <xsd:attribute name="Flora" type="xsd:string"/>
        <xsd:attribute name="Stock" type="xsd:string"/>
        <xsd:attribute name="Toy" type="xsd:string"/>
        <xsd:attribute name="Sprinklered" type="xsd:string"/>
        <xsd:attribute name="Luggage" type="xsd:string"/>
    </xsd:complexType>
    <xsd:element name="authors" type="xsd:string">
        <xsd:annotation>
            <xsd:documentation>Authors of publication article </xsd:documentation>
        </xsd:annotation>
    </xsd:element>
    <xsd:complexType name="dataType">
        <xsd:sequence>
            <xsd:element name="time" type="timeType"/>
            <xsd:element name="rhr" type="rhrType"/>
        </xsd:sequence>
        <xsd:attribute name="type">
            <xsd:simpleType>
                <xsd:restriction base="xsd:NMTOKEN">
                    <xsd:enumeration value="Attr"/>
                    <xsd:enumeration value="Rhr"/>
                </xsd:restriction>
            </xsd:simpleType>
```

```
    </xsd:attribute>
    <xsd:attribute name="type" type="xsd:string"/>
</xsd:complexType>
<xsd:complexType name="descriptionType">
    <xsd:annotation>
        <xsd:documentation>Description of item</xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
        <xsd:element ref="summary"/>
        <xsd:element ref="details" minOccurs="0"/>
    </xsd:sequence>
</xsd:complexType>
<xsd:element name="details" type="xsd:string">
    <xsd:annotation>
        <xsd:documentation>Multi-line description of the record</xsd:documentation>
    </xsd:annotation>
</xsd:element>
<xsd:element name="document" type="xsd:string">
    <xsd:annotation>
        <xsd:documentation>Name of the publication</xsd:documentation>
    </xsd:annotation>
</xsd:element>
<xsd:element name="FireBaseXML">
    <xsd:annotation>
        <xsd:documentation>Document element</xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
        <xsd:sequence>
            <xsd:element ref="version"/>
            <xsd:element ref="date"/>
            <xsd:element name="author" type="authorType"/>
            <xsd:element ref="description"/>
            <xsd:element name="item" type="itemType" maxOccurs="unbounded"/>
        </xsd:sequence>
        <xsd:attribute name="hrrs" type="xsd:string" use="required"/>
    </xsd:complexType>
</xsd:element>
<xsd:complexType name="heat_of_combustionType">
    <xsd:annotation>
        <xsd:documentation>Average heat of combustion of test item</xsd:documentation>
    </xsd:annotation>
    <xsd:simpleContent>
        <xsd:restriction base="xsd:float">
            <xsd:attribute name="units" use="required">
                <xsd:simpleType>
                    <xsd:restriction base="xsd:string">
                        <xsd:enumeration value="J/kg"/>
                        <xsd:enumeration value="kJ/kg"/>
                    </xsd:restriction>
```

```
                </xsd:simpleType>
            </xsd:attribute>
            <xsd:attribute name="type" type="xsd:string" fixed="available_energy"/>
        </xsd:restriction>
    </xsd:simpleContent>
</xsd:complexType>
<xsd:element name="identifier" type="xsd:string">
    <xsd:annotation>
        <xsd:documentation>Unique ID of record</xsd:documentation>
    </xsd:annotation>
</xsd:element>
<xsd:complexType name="initial_massType">
    <xsd:annotation>
        <xsd:documentation>Initial mass of test item</xsd:documentation>
    </xsd:annotation>
    <xsd:simpleContent>
        <xsd:restriction base="xsd:float">
            <xsd:attribute name="units" use="required">
                <xsd:simpleType>
                    <xsd:restriction base="xsd:string">
                        <xsd:enumeration value="kg"/>
                        <xsd:enumeration value="g"/>
                    </xsd:restriction>
                </xsd:simpleType>
            </xsd:attribute>
            <xsd:attribute name="type" type="xsd:string" fixed="mass"/>
        </xsd:restriction>
    </xsd:simpleContent>
</xsd:complexType>
<xsd:complexType name="itemType">
    <xsd:annotation>
        <xsd:documentation>An individual item in the database</xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
        <xsd:element name="attribute" type="attributeType"/>
        <xsd:element name="reference" type="referenceType" maxOccurs="unbounded"/>
        <xsd:element name="description" type="descriptionType"/>
        <xsd:element name="heat_of_combustion" type="heat_of_combustionType" minOccurs="0"/>
        <xsd:element name="initial_mass" type="initial_massType" minOccurs="0"/>
        <xsd:element name="data" type="dataType"/>
    </xsd:sequence>
    <xsd:attribute name="id" type="xsd:ID" use="required"/>
    <xsd:attribute name="type" use="required">
        <xsd:simpleType>
            <xsd:restriction base="xsd:string">
                <xsd:enumeration value="discrete"/>
            </xsd:restriction>
        </xsd:simpleType>
    </xsd:attribute>
```

```
</xsd:complexType>
<xsd:complexType name="linkType">
    <xsd:annotation>
        <xsd:documentation>Storage for link information</xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
        <xsd:element ref="URL"/>
        <xsd:element ref="page" minOccurs="0"/>
    </xsd:sequence>
    <xsd:attribute name="type" use="optional">
        <xsd:simpleType>
            <xsd:restriction base="xsd:string">
                <xsd:enumeration value="pdf"/>
                <xsd:enumeration value="web-page"/>
            </xsd:restriction>
        </xsd:simpleType>
    </xsd:attribute>
</xsd:complexType>
<xsd:element name="page" type="xsd:positiveInteger">
    <xsd:annotation>
        <xsd:documentation>Page number within online publication</xsd:documentation>
    </xsd:annotation>
</xsd:element>
<xsd:complexType name="referenceType">
    <xsd:annotation>
        <xsd:documentation>Source reference for this item</xsd:documentation>
    </xsd:annotation>
    <xsd:sequence>
        <xsd:element ref="title" minOccurs="0"/>
        <xsd:element ref="document" minOccurs="0"/>
        <xsd:element ref="authors" minOccurs="0"/>
        <xsd:element name="link" type="linkType" minOccurs="0"/>
    </xsd:sequence>
    <xsd:attribute name="type" type="xsd:string"/>
    <xsd:attribute name="source">
        <xsd:simpleType>
            <xsd:restriction base="xsd:NMTOKEN">
                <xsd:enumeration value="primary"/>
                <xsd:enumeration value="secondary"/>
            </xsd:restriction>
        </xsd:simpleType>
    </xsd:attribute>
    <xsd:attribute name="id" type="xsd:ID"/>
    <xsd:attribute name="organisation" type="xsd:string"/>
    <xsd:attribute name="idref" type="xsd:IDREF"/>
</xsd:complexType>
<xsd:complexType name="rhrType">
    <xsd:annotation>
        <xsd:documentation>Energy flow data</xsd:documentation>
```

```
    </xsd:annotation>
    <xsd:simpleContent>
        <xsd:restriction base="xsd:string">
            <xsd:attribute name="units" use="required">
                <xsd:simpleType>
                    <xsd:restriction base="xsd:string">
                        <xsd:enumeration value="kW"/>
                    </xsd:restriction>
                </xsd:simpleType>
            </xsd:attribute>
            <xsd:attribute name="type" type="xsd:string" fixed="heat_flow_rate"/>
        </xsd:restriction>
    </xsd:simpleContent>
</xsd:complexType>
<xsd:element name="summary" type="xsd:string">
    <xsd:annotation>
        <xsd:documentation>One-line description of the item</xsd:documentation>
    </xsd:annotation>
</xsd:element>
<xsd:complexType name="timeType">
    <xsd:annotation>
        <xsd:documentation>Time data</xsd:documentation>
    </xsd:annotation>
    <xsd:simpleContent>
        <xsd:restriction base="xsd:string">
            <xsd:attribute name="units" use="required">
                <xsd:simpleType>
                    <xsd:restriction base="xsd:string">
                        <xsd:enumeration value="s"/>
                    </xsd:restriction>
                </xsd:simpleType>
            </xsd:attribute>
            <xsd:attribute name="type" type="xsd:string" fixed="time"/>
        </xsd:restriction>
    </xsd:simpleContent>
</xsd:complexType>
<xsd:element name="title">
    <xsd:annotation>
        <xsd:documentation>Title of publication article</xsd:documentation>
    </xsd:annotation>
    <xsd:complexType>
        <xsd:simpleContent>
            <xsd:extension base="xsd:string"/>
        </xsd:simpleContent>
    </xsd:complexType>
</xsd:element>
<xsd:element name="URL" type="xsd:anyURI">
    <xsd:annotation>
        <xsd:documentation>A URL to an online publication</xsd:documentation>
```

```
        </xsd:annotation>
    </xsd:element>
    <xsd:complexType name="authorType">
        <xsd:annotation>
            <xsd:documentation>Author of this database</xsd:documentation>
        </xsd:annotation>
        <xsd:simpleContent>
            <xsd:restriction base="xsd:string">
                <xsd:attribute name="email" type="xsd:string" use="optional"/>
            </xsd:restriction>
        </xsd:simpleContent>
    </xsd:complexType>
    <xsd:element name="description" type="xsd:string">
        <xsd:annotation>
            <xsd:documentation>Description of database</xsd:documentation>
        </xsd:annotation>
    </xsd:element>
    <xsd:element name="version" type="xsd:decimal">
        <xsd:annotation>
            <xsd:documentation>Version number of this database</xsd:documentation>
        </xsd:annotation>
    </xsd:element>
    <xsd:element name="date" type="xsd:date">
        <xsd:annotation>
            <xsd:documentation>Version date of this database</xsd:documentation>
        </xsd:annotation>
    </xsd:element>
</xsd:schema>
```

APPENDIX C. WEB-BASED INTERFACE SCRIPTS TO FIREBASEXML DATABASES

C.1 Source listing for viewTest.html file (version 2003.1)

```
1:  <html>
2:      <head>
3:          <title>View test details</title>
4:          <link rel="stylesheet" type="text/css" href="instructions.css" />
5:      </head>

        <!-- Declare database and stylesheet -->
6:      <XML id="source"></XML>
7:      <XML id="style" src="viewTest.xslt"></XML>

        <!-- Perform transformation -->
8:      <SCRIPT FOR="window" EVENT="onload" language="jscript">
9:          var node;

            // Get the search parameters of the calling document and split by '&'s
10:         list = document.location.search.split("&");

            // Find database name from first parameter in the list and load
11:         with (source) {
12:             async = false;
13:             db = list[0].split("=");
14:             path = db[1];
15:             load(path);
16:             }

            // Find the test name from the second parameter and construct query
17:         test = list[1].split("=");
18:         query = "//item[@id = '" + test[1] + "']";

            // Get node and transform
19:         node = source.selectSingleNode(query);
20:         htmlObj.innerHTML = node.transformNode(style);

21:     </SCRIPT>

22:     <body>
23:         <DIV ID="htmlObj"></DIV>
24:         <hr align="center"/>
25:     </body>
26:  </html>
```

C.2 Source listing for viewTest.xslt (version 2004.1)

```
1:    <?xml version="1.0" encoding="UTF-8"?>
2:    <xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

          <!-- root element template -->
3:        <xsl:template match="/">
4:            <xsl:apply-templates select="FireBaseXML"/>
5:        </xsl:template>

          <!--
************************************************************************************
-->
          <!-- <FireBaseXML> element template -->
6:        <xsl:template match="FireBaseXML">
7:            <xsl:apply-templates select="item"/>
8:        </xsl:template>

          <!--
************************************************************************************
-->
          <!-- <item> element template -->
9:        <xsl:template match="item">
10:           <xsl:apply-templates select="@id"/>
11:           <xsl:apply-templates select="description"/>
12:           <xsl:apply-templates select="reference"/>
13:           <xsl:apply-templates select="initial_mass"/>
14:           <xsl:apply-templates select="heat_of_combustion"/>
15:           <xsl:apply-templates select="data"/>
16:       </xsl:template>

          <!--
************************************************************************************
-->
          <!-- id attribute template -->
17:       <xsl:template match='@id'>
18:           <h1><xsl:value-of select="."/></h1>
19:       </xsl:template>

          <!--
************************************************************************************
-->
          <!-- <description> element template -->
20:       <xsl:template match='description'>
21:           <xsl:apply-templates select="summary"/>
22:           <xsl:apply-templates select="details"/>
23:       </xsl:template>

          <!--
************************************************************************************
-->
          <!-- <summary> element template -->
24:       <xsl:template match='summary'>
```

```
25:             <p><b>Description summary: </b><xsl:value-of select="."/></p>
26:         </xsl:template>

            <!--
*************************************************************************************
-->
            <!-- <details> element template -->
27:         <xsl:template match='details'>
28:             <p><b>Details: </b><xsl:value-of select="."/></p>
29:         </xsl:template>

            <!--
*************************************************************************************
***
                    <reference> element -->
30:         <xsl:template match='reference'>

31:             <xsl:if test="not(@idref)">
32:                 <p>
33:                 <b><xsl:text>Reference: </xsl:text></b>
34:                 <xsl:apply-templates select="authors"/>
35:                 <xsl:apply-templates select="title"/>
36:                 <xsl:apply-templates select="document"/>
37:                 <a>
38:                 <xsl:attribute name="href"><xsl:value-of select="link/URL"/>
                    </xsl:attribute>
39:                 <xsl:value-of select="link/URL"/>
40:                 </a>
41:                 </p>
42:             </xsl:if>

43:             <xsl:if test="@idref">
44:                 <xsl:apply-templates select="@idref"/>
45:             </xsl:if>
46:         </xsl:template>

            <!--
*************************************************************************************
-->
            <!-- idref attribute template -->
47:         <xsl:template match='@idref'>
48:             <xsl:variable name="ref" select="."/>
49:             <xsl:apply-templates select='//*[./@id=$ref]'/>
50:         </xsl:template>

            <!--
*************************************************************************************
-->
            <!-- <authors> element template -->
51:         <xsl:template match='authors'>
52:             <xsl:value-of select="."/>.
53:         </xsl:template>
```

```
        <!--
*****************************************************************************************
-->
        <!-- <title> element template -->
54:     <xsl:template match='title'>
55:         <xsl:value-of select="."/>.
56:     </xsl:template>

        <!--
*****************************************************************************************
-->
        <!-- <document> element template -->
57:     <xsl:template match='document'>
58:         <xsl:value-of select="."/>.
59:     </xsl:template>

        <!--
*****************************************************************************************
-->
        <!-- <initial_mass> element template -->
60:     <xsl:template match='initial_mass'>
61:         <p><b>Initial mass: </b>
62:         <xsl:value-of select="."/>&#32;
63:         <xsl:value-of select="@units"/>
64:         </p>
65:     </xsl:template>

        <!--
*****************************************************************************************
-->
        <!-- <heat_of_combustion> element template -->
66:     <xsl:template match='heat_of_combustion'>
67:         <p><b>Heat of combustion: </b>
68:         <xsl:value-of select="."/>&#32;
69:         <xsl:value-of select="@units"/>
70:         </p>
71:     </xsl:template>

        <!--
*****************************************************************************************
***
                <data> element -->
72:     <xsl:template match='data'>
73:         <xsl:variable name="csv-time" select="time"/>
74:         <xsl:variable name="csv-rhr" select="rhr"/>

75:         <xsl:if test="string-length($csv-time)>0">
76:             <p><b>Data:</b></p>
77:             <table>
78:             <thead>
79:             <td align='right'><i>Time (<xsl:value-of select="time/@units"/>)</i></td>
80:             <td align='right'><i>Rhr (<xsl:value-of select="rhr/@units"/>)</i></td>
81:             </thead>
82:             <xsl:call-template name='decompose'>
```

```
83:                 <xsl:with-param name="t"><xsl:value-of select="$csv-time"/>
                        </xsl:with-param>
84:                 <xsl:with-param name="r"><xsl:value-of select="$csv-rhr"/>
                        </xsl:with-param>
85:             </xsl:call-template>
86:             </table>
87:         </xsl:if>

88:     </xsl:template>

        <!--
*****************************************************************************
****
                decompose template -->
89:     <xsl:template name='decompose'>
90:         <xsl:param name="t"></xsl:param>
91:         <xsl:param name="r"></xsl:param>

92:         <tr>
93:         <xsl:choose>

                <!-- Check for a comma in the time parameter string -->
94:             <xsl:when test="contains($t, ',')">

                    <!-- A comma has been found, decompose lists and iterate -->
95:                 <xsl:variable name="csv-time" select="substring-after($t,',')"/>
96:                 <td align='right'><xsl:value-of select="substring-before($t,',')"/></td>
97:                 <xsl:variable name="csv-rhr" select="substring-after($r,',')"/>
98:                 <td align='right'><xsl:value-of select="substring-before($r,',')"/></td>

99:                 <xsl:if test="string-length($csv-time)>0">
100:                    <xsl:call-template name='decompose'>
101:                        <xsl:with-param name="t"><xsl:value-of select="$csv-time"/>
                                </xsl:with-param>
102:                        <xsl:with-param name="r"><xsl:value-of select="$csv-rhr"/>
                                </xsl:with-param>
103:                    </xsl:call-template>
104:                </xsl:if>

105:            </xsl:when>

                <!-- No comma found so we must be at the last value in our csv lists -->
106:            <xsl:otherwise>
107:                <td align='right'><xsl:value-of select="$t"/></td>
108:                <td align='right'><xsl:value-of select="$r"/></td>
109:            </xsl:otherwise>

110:        </xsl:choose>
111:        </tr>

112:    </xsl:template>
```

```
        <!--
*****************************************************************************
-->
113:    </xsl:stylesheet>
```

APPENDIX D. BRANZFIRE TRANSFORMATIONS FOR A FIREBASEXML DATABASE

D.1 The BRANZFIRE XSL transformation document (hrrt_branzfire.xslt, version 2004.1)

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="hrrt_viewConversion.xslt"?>
<!--
*   h r r t _ b r a n z f i r e . x s l t
*   ========================
*
*   Stylesheet to transform a FireBaseXML record to a format suitable for BRANZFire.
*
*   Mike Spearpoint
*   07 Jan 2004
*   version 2004.1
-->
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
    <xsl:output omit-xml-declaration='yes' encoding='UTF-8' indent='yes' method='xml'/>

    <hrrt:Transformation xmlns:hrrt="http://www.civil.canterbury.ac.nz/spearpoint/
            HRR_Database/FireBaseXML_Conversion.xsd">

        <hrrt:name>BRANZFire</hrrt:name>
        <hrrt:version>2004.1</hrrt:version>

        <hrrt:description language="english">
            <hrrt:summary>FireBaseXML interface to BRANZFire.</hrrt:summary>
            <hrrt:details>Converts a FireBaseXML database record for import into
                BRANZFire.</hrrt:details>
        </hrrt:description>

        <hrrt:author email="m.spearpoint@civil.canterbury.ac.nz">
                Michael Spearpoint</hrrt:author>
        <hrrt:output>
            <hrrt:extension>xml</hrrt:extension>
        </hrrt:output>

    </hrrt:Transformation>

<!--
*****************************************************************************************
-->
    <!-- root element -->
    <xsl:template match="/">
        <xsl:apply-templates select="item"/>
    </xsl:template>

<!--
*****************************************************************************************
-->
    <!-- <item> element -->
    <xsl:template match="item">
```

```
        <xsl:element name="BRANZFire-XML">
            <xsl:apply-templates select="data"/>
            <xsl:apply-templates select="heat_of_combustion"/>
            <xsl:apply-templates select="attribute"/>
            <xsl:apply-templates select="description"/>
        </xsl:element>
    </xsl:template>

<!--
*****************************************************************************************
-->
    <!-- <heat-of-combustion> element -->
    <xsl:template match='heat_of_combustion'>

        <xsl:element name="heat-of-combustion">
            <xsl:value-of select="."/>
        </xsl:element>

    </xsl:template>

<!--
*****************************************************************************************
-->
    <!-- <attribute> element -->
    <xsl:template match='attribute'>

        <!-- Create the <object-type> node and
             insert the name of the first attribute -->
        <xsl:element name="object-type">
            <xsl:value-of select="name(@*)"/>
        </xsl:element>

    </xsl:template>

<!--
*****************************************************************************************
-->
    <!-- <description> element -->
    <xsl:template match='description'>

        <xsl:element name="description">
            <xsl:value-of select="./summary"/>
            <xsl:text> - </xsl:text>
            <xsl:value-of select="./details"/>
        </xsl:element>

    </xsl:template>

<!--
*****************************************************************************************
-->
    <!-- <data> element -->
    <xsl:template match='data'>

            <xsl:variable name="csv-time" select="time"/>
            <xsl:variable name="csv-rhr" select="rhr"/>

            <!-- Output units -->
            <xsl:element name="units">
                <xsl:value-of select="time/@units"/>
```

```
            <xsl:text>,</xsl:text>
            <xsl:value-of select="rhr/@units"/>
            <xsl:text>&#xA;</xsl:text>
        </xsl:element>

        <!-- Extract data -->
        <xsl:if test="string-length($csv-time)>0">
            <xsl:element name="tab-data">
                <xsl:call-template name='decompose'>
                    <xsl:with-param name="t"><xsl:value-of select="$csv-time"/>
                        </xsl:with-param>
                    <xsl:with-param name="r"><xsl:value-of select="$csv-rhr"/>
                        </xsl:with-param>
                </xsl:call-template>
            </xsl:element>
        </xsl:if>

    </xsl:template>

<!--
*****************************************************************************
-->
    <!-- decompose template -->
    <xsl:template name='decompose'>
        <xsl:param name="t"></xsl:param>
        <xsl:param name="r"></xsl:param>


        <xsl:choose>

            <!-- Check for a comma in the time parameter string -->
            <xsl:when test="contains($t, ',')">

                <!-- A comma has been found, decompose lists and iterate -->
                <xsl:variable name="csv-time" select="substring-after($t,',')"/>
                <xsl:variable name="time" select="substring-before($t,',')"/>
                <xsl:value-of select="format-number($time, '0.0')"/>
                <xsl:text>,</xsl:text>

                <xsl:variable name="csv-rhr" select="substring-after($r,',')"/>
                <xsl:variable name="rhr" select="substring-before($r,',')"/>
                <xsl:value-of select="format-number($rhr, '0.0')"/>
                <xsl:text>&#x09;</xsl:text>

                <xsl:if test="string-length($csv-time)>0">
                    <xsl:call-template name='decompose'>
                        <xsl:with-param name="t"><xsl:value-of select="$csv-time"/>
                            </xsl:with-param>
                        <xsl:with-param name="r"><xsl:value-of select="$csv-rhr"/>
                            </xsl:with-param>
                    </xsl:call-template>
                </xsl:if>

            </xsl:when>

            <!-- No comma found so we must be at the last value in our csv lists -->
            <xsl:otherwise>
                <xsl:value-of select="format-number($t, '0.0')"/>
                <xsl:text>,</xsl:text>
                <xsl:value-of select="format-number($r, '0.0')"/>
```

```
            <xsl:text>&#xA;</xsl:text>
        </xsl:otherwise>

    </xsl:choose>


</xsl:template>

<!--
********************************************************************************
-->

</xsl:stylesheet>
```

D.2 XML document generated by the BRANZFIRE XSL transformation

The output from the hrrt_branzfire.xslt conversion using the example FireBaseXML record (Figure 6-17) is

```
<BRANZFire-XML>
    <units>s,kW</units>
    <tab-data>0.0,0.0 155.0,80.0 187.0,200.0 212.0,365.0 250.0,1860.0 260.0,1940.0
    270.0,1930.0 280.0,1945.0 300.0,1500.0 360.0,500.0 420.0,250.0 512.0,200.0
    675.0,250.0 840.0,160.0 1200.0,115.0</tab-data>
    <heat-of-combustion>18100</heat-of-combustion>
    <object-type>Armchair</object-type>
    <description>Easy chair, F21 - Wood frame easy chair, polyeurathane padding,
        polyolefin fabric.</description>
</BRANZFire-XML>
```

D.3 The BRANZFIRE `frmFireData.Form_Load()` procedure

```
Private Sub Form_Load()
'*  ===========================================================
'*
'*      This event centres the form on the screen
'*      when the form first loads.
'*
'*      FireBaseXML modifications 30 Oct 2001 Mike Spearpoint
'*
'*  ===========================================================


    ' Specify the FireBaseXML online database URL and
    ' appropriate XSLT file
    FireBaseXMLIxCtrl.setDatabaseURL
("http://www.civil.canterbury.ac.nz/spearpoint/HRR_Database/HRR
_Database.xml")
    FireBaseXMLIxCtrl.setTransformationURL
("http://www.civil.canterbury.ac.nz/spearpoint/HRR_Database/hrr
t_branzfire.xslt")


    'call procedure to centre form on screen
    Centre_Form Me
    On Error Resume Next
    frmFireData.datPrimaryRS.DatabaseName = FireDatabaseName
    frmFireData.datPrimaryRS.RecordSource = "Fire Data"
    frmFireData.datPrimaryRS.Refresh
    plot_graph

End Sub
```

D.4 The BRANZFIRE frmFireData.FireBaseXMLIxCtrl_Data Ready() procedure

```
'*************************************************************
'*
'*  F i r e B a s e X M L I x C t r l _ D a t a R e a d y
'*  ==================================================
'*
'*  Respond to the DataReady event from the FireBaseXMLIxCtrl
'*  by transferring data into appropriate fields.
'*
'*  Added 30 Oct 2001 Mike Spearpoint
'*
'*************************************************************
Private Sub FireBaseXMLIxCtrl_DataReady()

    Dim hoc As Double


    ' Update data text boxes
    txtDBHRR.Text = FireBaseXMLIxCtrl.getTabdata
    txtFields(7) = FireBaseXMLIxCtrl.getDescription
    Combo1 = FireBaseXMLIxCtrl.getObjectType

    ' Update heat of combustion
    hoc = FireBaseXMLIxCtrl.getHeatOfCombustion
    If hoc > 0 Then txtfields(0) = hoc Else txtfields(0) = ""

    ' Blank out undefined fields
    txtFields(2) = ""
    txtFields(4) = ""
    txtFields(5) = ""
    txtFields(9) = ""

    ' Update graph
    Call cmdUpdate_Click

End Sub
```

D.5 FireBaseXMLIx Control help documentation

FireBaseXMLIx Control

A basic interface to a FireBaseXML database

Syntax

FireBaseXMLIx

DataReady Event

Occurs when the data is ready after a user has selected to extract a record from the database.

Syntax

Private Sub *object*_DataReady ()

---

setDatabaseURL Method

Sets the URL of the FireBaseXML database.

Syntax

object.**setDatabaseURL(***url* **as String) as Boolean**

The **setDatabaseURL** method syntax has these parts:

Part	Description
url	The full URL of the database.

Returns

The method returns **True** if successful.

Remarks

The method does not check the database exists or open the database.

---

setTransformationURL Method

Sets the URL of the XSL transformation document that is to be applied to a selected test record extracted from a FireBaseXML database.

Syntax

object.**setTransformationURL(***url* **as String) as Boolean**

The **setTransformationURL** method syntax has these parts:

Part	Description
url	The full URL of the transformation.

Returns

The method returns **True** if successful.

Remarks

The method does not check the transformation exists or read the transformation document.

Error messages

1001	Selected transformation unavailable
1002	No transformations available
1003	Failed to open selected database

D.6 FireBaseXMLIxBF Control help documentation

getDescription Method

Gets the text description of a selected test record.

Syntax

object.**getDescription() as String**

Returns

The method returns a text string that describes the selected test record.

---

getTabdata Method

Gets the rate of heat release data for selected test record.

Syntax

object.**getTabdata() as String**

Returns

The method returns a text string that is the rate of heat release data.

Remarks

The text is formatted with commas separating each time and heat release field and Tabs separating each data pair. This format allows direct import into the appropriate text box in the BRANZFire Fire Object Database window.

getObjectType Method

Gets the object type description of a selected test record.

Syntax

object.**getObjectType() as String**

Returns

The method returns a text string that describes the object type of the selected test record.

---

getHeatOfCombustion Method

Gets the heat of combustion of a selected test record.

Syntax

object.**getHeatOfCombustion() as Double**

Returns

The method returns the heat of combustion (in kJ/g) of a selected test record. A value of zero is returned if no heat of combustion exists in the selected test.

APPENDIX E. STEPPARSER-DATA-CONSTRUCTION SCHEMA

E.1 The complete STEPParser-data-construction schema

```
<?xml version="1.0" encoding="UTF-8"?>
<!--W3C Schema generated by XML Spy v4.2 U (http://www.xmlspy.com)-->
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
elementFormDefault="qualified">
    <xs:element name="STEPParser-data-construction">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="project"/>
                <xs:element ref="space" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="version" type="xs:decimal" use="required"/>
            <xs:attribute name="author" type="xs:string" use="required"/>
            <xs:attribute name="organisation" type="xs:string" use="required"/>
            <xs:attribute name="created" type="xs:string" use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="door">
        <xs:complexType>
            <xs:attribute name="id" type="xs:string" use="required"/>
            <xs:attribute name="name" type="xs:string" use="required"/>
            <xs:attribute name="width" type="xs:long" use="required"/>
            <xs:attribute name="height" type="xs:long" use="required"/>
            <xs:attribute name="pos_x" type="xs:long" use="required"/>
            <xs:attribute name="pos_y" type="xs:long" use="required"/>
            <xs:attribute name="pos_z" type="xs:long" use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="material">
        <xs:complexType>
            <xs:attribute name="id" type="xs:string" use="required"/>
            <xs:attribute name="description" type="xs:string" use="required"/>
            <xs:attribute name="thickness" type="xs:long" use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="materials">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="material" maxOccurs="unbounded"/>
            </xs:sequence>
            <xs:attribute name="id" type="xs:string" use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="opening">
        <xs:complexType>
            <xs:attribute name="id" type="xs:string" use="required"/>
            <xs:attribute name="name" type="xs:string" use="required"/>
            <xs:attribute name="width" type="xs:long" use="required"/>
            <xs:attribute name="soffit" type="xs:long" use="required"/>
```

```
        <xs:attribute name="sill" type="xs:long" use="required"/>
        <xs:attribute name="pos_x" type="xs:long" use="required"/>
        <xs:attribute name="pos_y" type="xs:long" use="required"/>
        <xs:attribute name="pos_z" type="xs:long" use="required"/>
    </xs:complexType>
</xs:element>
<xs:element name="project">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="units"/>
        </xs:sequence>
        <xs:attribute name="id" type="xs:string" use="required"/>
        <xs:attribute name="name" type="xs:string" use="required"/>
    </xs:complexType>
</xs:element>
<xs:element name="slab">
    <xs:complexType>
        <xs:sequence>
            <xs:element ref="materials"/>
        </xs:sequence>
        <xs:attribute name="id" type="xs:string" use="required"/>
        <xs:attribute name="name" type="xs:string" use="required"/>
        <xs:attribute name="type" use="required">
            <xs:simpleType>
                <xs:restriction base="xs:NMTOKEN">
                    <xs:enumeration value="FLOOR"/>
                    <xs:enumeration value="ROOF"/>
                </xs:restriction>
            </xs:simpleType>
        </xs:attribute>
        <xs:attribute name="pos_x" type="xs:long" use="required"/>
        <xs:attribute name="pos_y" type="xs:long" use="required"/>
        <xs:attribute name="pos_z" type="xs:long" use="required"/>
    </xs:complexType>
</xs:element>
<xs:element name="space">
    <xs:complexType>
        <xs:choice maxOccurs="unbounded">
            <xs:element ref="window"/>
            <xs:element ref="door"/>
            <xs:element ref="wall"/>
            <xs:element ref="slab"/>
            <xs:element ref="opening"/>
        </xs:choice>
        <xs:attribute name="id" type="xs:string" use="required"/>
        <xs:attribute name="name" type="xs:string" use="required"/>
        <xs:attribute name="width" type="xs:long" use="required"/>
        <xs:attribute name="depth" type="xs:long" use="required"/>
        <xs:attribute name="height" type="xs:long" use="required"/>
        <xs:attribute name="pos_x" type="xs:long" use="required"/>
        <xs:attribute name="pos_y" type="xs:long" use="required"/>
        <xs:attribute name="pos_z" type="xs:long" use="required"/>
    </xs:complexType>
</xs:element>
<xs:element name="units">
```

```
        <xs:complexType>
            <xs:attribute name="id" type="xs:string" use="required"/>
            <xs:attribute name="length" type="xs:string" use="required"/>
            <xs:attribute name="area" type="xs:string" use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="wall">
        <xs:complexType>
            <xs:sequence>
                <xs:element ref="materials"/>
            </xs:sequence>
            <xs:attribute name="id" type="xs:string" use="required"/>
            <xs:attribute name="name" type="xs:string" use="required"/>
            <xs:attribute name="pos_x" type="xs:long" use="required"/>
            <xs:attribute name="pos_y" type="xs:long" use="required"/>
            <xs:attribute name="pos_z" type="xs:long" use="required"/>
        </xs:complexType>
    </xs:element>
    <xs:element name="window">
        <xs:complexType>
            <xs:attribute name="id" type="xs:string" use="required"/>
            <xs:attribute name="name" type="xs:string" use="required"/>
            <xs:attribute name="width" type="xs:long" use="required"/>
            <xs:attribute name="soffit" type="xs:long" use="required"/>
            <xs:attribute name="sill" type="xs:long" use="required"/>
            <xs:attribute name="pos_x" type="xs:long" use="required"/>
            <xs:attribute name="pos_y" type="xs:long" use="required"/>
            <xs:attribute name="pos_z" type="xs:long" use="required"/>
        </xs:complexType>
    </xs:element>
</xs:schema>
```